FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Job Operation Software
End-user's Guide

J2UL-2534-01ENZ0(09)
March 2024

Preface

Purpose of This Manual

This manual describes the job operation procedures of "Job Operation Software" included in Technical Computing Suite.

Intended Readers

This manual is intended the administrators who operate and manage jobs with Job Operation Software, and the end-user to actually operate job.

The manual assumes that readers as follows.

- Linux basic knowledge is required.
- Understand the overview of Job Operation Software in the "Job Operation Software Overview."
- Knowledge of container virtualization technology and Docker in Linux.
- Knowledge of Linux virtualization technology (KVM) and virtualization management utilities (libvirt and virsh)

Organization of This Manual

This manual is organized as follows.

Chapter 1 Job Mechanism

This chapter describes the mechanism of jobs.

Chapter 2 Job Operation Procedures

This chapter describes job operation procedures.

Appendix A Job Information

This appendix describes the outputs of the command concerning information on the job.

Appendix B Notification Message Related to Job Execution

This appendix describes the messages that are notified to user by e-mail, if a job terminates abnormally.

Appendix C Executing programs of MPI processing system other than Development Studio

This appendix describes how to execute MPI programs of the MPI processing system other than Development Studio on the Job Operation Software and the notes on executing them.

Appendix D Operations on Jobs

This appendix describes the relationship between job states and possibility of operations on jobs.

Appendix E Using Job Execution Environment

This appendix describes how to create an image file for a job execution environment and troubleshooting for using job execution environment.

Notation Used in This Manual

Notation of users

The users of the Job Operation Software include the administrators responsible for system management and job operations and the end users who use the system to run programs. Unless otherwise noted, "user" in this manual means an end user.

Representation of units

The following table lists the prefixes used to represent units in this manual. Basically, disk size is represented as a power of 10, and memory size is represented as a power of 2. Be careful about specifying them when displaying or entering commands.

Prefix	Value	Prefix	Value
K (kilo)	10^3	Ki (kibi)	2^{10}
M (mega)	10^6	Mi (mebi)	2^{20}

Prefix	Value	Prefix	Value
G (giga)	10^9	Gi (gibi)	2^{30}
T (tera)	10^{12}	Ti (tebi)	2^{40}
P (peta)	10^{15}	Pi (pebi)	2^{50}

Notation of model names

In this manual, the computer that based on Fujitsu A64FX CPU is abbreviated as "FX server", and FUJITSU server PRIMERGY as "PRIMERGY server" (or simply "PRIMERGY").
Also, specifications of some of the functions described in the manual are different depending on the target model. In the description of such a function, the target model is represented by its abbreviation as follows:
[FX]: The description applies to FX servers.
[PG]: The description applies to PRIMERGY servers.

Path names of the commands

In the examples of the operations, the path names of the commands in the directory /bin, /usr/bin, /sbin or /usr/sbin might not be represented by absolute path.

Symbols in this manual

This manual uses the following symbols.

Note

The Note symbol indicates an item requiring special care. Be sure to read these items.

See

The See symbol indicates the written reference source of detailed information.

Information

The Information symbol indicates a reference note related to Job Operation Software.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Red Hat and Red Hat Enterprise Linux are registered trademarks of Red Hat, Inc. in the U.S. and other countries.
- Intel is trademark of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
- All other trademarks are the property of their respective owners.

Date of publication and Version

Version	Manual Code
March 2024, Version 1.9	J2UL-2534-01ENZ0(09)
March 2023, Version 1.8	J2UL-2534-01ENZ0(08)
March 2022, Version 1.7	J2UL-2534-01ENZ0(07)

Version	Manual Code
November 2021, Version 1.6	J2UL-2534-01ENZ0(06)
August 2021, Version 1.5	J2UL-2534-01ENZ0(05)
March 2021, Version 1.4	J2UL-2534-01ENZ0(04)
December 2020, Version 1.3	J2UL-2534-01ENZ0(03)
September 2020, Version 1.2	J2UL-2534-01ENZ0(02)
June 2020, Version 1.1	J2UL-2534-01ENZ0(01)
February 2020, First version	J2UL-2534-01ENZ0(00)

Update history

Changes	Location	Version
Added information that is output when the number of uncompleted files exceeds 1000.	2.3.3.10	1.9
Added a note that TCP/IP jobs should be run on no more than 512 nodes.	1.6.3.1	1.8
Improved the description of execution examples for MPI processing systems other than Development Studio.	C.1	
Added that job statistical information PERF COUNT is not output for interactive jobs.	A.2	1.7
Added description of job statistical information about node power consumption on FX server.		
Changed the calculating job performance information. - Number of memory read requests - Number of memory write requests	A.3	
Added notes about environment variables in jobs for use with the LLIO feature.	2.1.2	1.6
Added notes on collecting LLIO performance information.	2.3.3.9	
Changed the URL for McKernel information.	E.1.2	1.5
Support for the environment variable PLE_RANK_ON_NODE which is set for MPI processes.	2.3.6.10	1.4
Added to following descriptions for the McKernel mode. - A method for calculating the amount of job memory for McKernel and behavior when the memory for the host OS runs out. - Specifying McKernel boot parameters. And, added a note to the KVM mode description regarding job execution environments that enabled writing to virtual machine image files.	2.3.8 A.2	
Added description of UDI specification in McKernel mode.	1.9 2.3.8 E.1.2 E.2.2	1.3
Added a description of when shared temporary area running out of space. Corrected the minimum size per node required for the second-layer storage cache.	2.3.3.1	
Improved action for a notification message "Killed by OOM killer." related to job execution.	Appendix B	
Changed the URL for McKernel information.	1.9 E.1.2	1.2

Changes	Location	Version
Corrected the description of the disturbance to job performance due to the job statistical information collection process.	2.1.1.8	
Added an environment variable PJM_NODE_ALLOCATION_IO_MODE to be set in the job.	2.1.2	
Corrected notes for the llio_transfer command.	2.1.4	
Added the following parameters that the pjacl command outputs. - The parameter 'default allocation-io-mode' in the item 'defines'. - The parameter 'pjsub(no-io-exclusive)' and 'pjsub(io-exclusive)' of the item 'execute'.	2.2.2	
The I/O-exclusive mode and I/O-sharing mode are supported as node allocation methods for jobs.	1.2.6 2.3.2.11	
Added notes about job statistical information values for jobs in the McKernel mode and KVM mode.	A.2	
Moved descriptions of the startup configuration files that must be configured by the administrator to "Job Operation Software Administrator's Guide for Job Management".	E.1.1	
Changed how to obtain Red Hat documentation for KVM mode virtual machines.	E.1.3	
Improved the description of execution units for which the node temporary area and the shared temporary area are secured.	1.13	1.1
Added procedure for avoiding disturbance to job execution performance.	2.1.1.8	
Added a description about the size consumed for file management in the shared temporary area and the node temporary area.	2.3.3.1	
Added descriptions of the standard output and standard error output files for the mpiexec command.	2.3.2.10 2.6.1	
Added a description of the message "[INFO] PLE 0094" that is output when an interactive job exits after <Ctrl-c> has been entered.	2.3.4.5	
Added a description of recommended options for standard output and standard error output files when running large-scale MPI jobs.	2.3.6.9	
Added a description of the environment variable PMIX_RANK to be set in the MPI processes.	2.3.6.10	
Added a note for Docker mode that the container image for UDI specification must be referenced from compute nodes.	2.3.8	
Added a note for McKernel mode that when running MPI jobs, the user may need to increase the amount of memory allocated on the host OS.		
Added a note for KVM mode that the virtual machine image file for UDI specification must be referenced from compute nodes and the job must be submitted under the home directory.		
Added a note about what job status the job job parameters can be changed with the pjalter command. Added a note about using the pjalter command to change the resource unit or the resource group on which the job runs.	2.5.4	
Fixed the job state and description of action when the message "Reason: Node down." is notified.	Appendix B	
Added required an OS package and mount points for LLIO in creating Docker mode image file.	E.1.1	
Added references to information about the virtual machine for KVM mode.	E.1.3	
Added 'Connection that is not associated with the device name' as a requirement for virtual machines in KVM mode.		
Improved the description to procure the resource management agent package and the source package of the agent required to use KVM mode.		

Contents

Chapter 1 Job Mechanism

The Job Operation Software processes programs, created by users, in units called "jobs." This chapter describes jobs.

1.1 What Are Jobs?

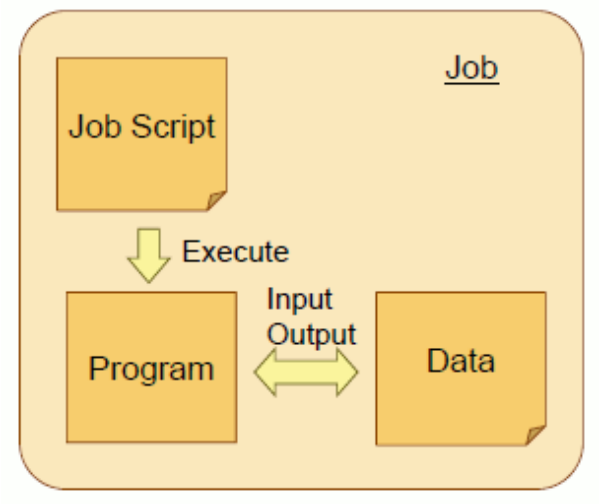
On the system where the Job Operation Software is installed, users do not execute programs directly. Instead, they request the job operation management function of the Job Operation Software to execute programs. Upon receiving a program execution request, the job operation management function reserves the computer resources required for executing the program, and executes it.

The unit of processing of such a program is a job. A job consists of a program, its input/output data, and the job script with the written program execution procedure.

Table 1.1 Job components

Component	Description
Program	A program is an executable program prepared by a user. An example is an execution file compiled by Development Studio, which is related software.
Data	Data is program inputs and outputs. Examples include a data file that stores input parameters for the program, an output file that stores processing results, and program display results.
Job script	A job script is a shell script with the written program execution procedure. The Job Operation Software executes a job based on this job script.

Figure 1.1 Job configuration



The workflow for job execution is as follows.

1. The user prepares the files required for executing a job and places them on the login node.
2. The user requests the job operation management function to execute the job. This is called "job submission."

   The job is submitted by the pjsub command specifying the job script.
3. The job operation management function allocates computer resources to the job and executes the job script.
4. The function executes the program according to the contents of the job script, and outputs the execution results.
5. The user is notified by e-mail (only if the job submission includes an e-mail notification instruction) when the job script ends.
6. The user checks the job execution results on the login node.

See

The above description covers the general workflow for job execution. For details, see "Chapter 2 Job Operation Procedures."

1.2 Types of Job

Jobs are classified into several types, such as according to the required computer resource or the program type.

Jobs in the Job Operation Software are classified into the following types.

Table 1.2 Types of job

Classification	Type of job
Classification by job structure (job model)	Normal job
	Bulk job
	Step job
	Workflow job
	Master-Worker job
Classification by job execution format (job type)	Batch job
	Interactive job
Classification by parallelism	Sequential job
	Parallel job
Classification by required computer resources (node resources)	Single node job
	Multinode job
Classification by node resource allocation method	Node allocated job (Node-exclusive job)
	Virtual node allocated job (Node-exclusive job, Node-sharing job)
Classification by I/O node allocation method [FX]	I/O-exclusive job
	I/O-sharing job

Note

Unless otherwise noted, "node" in this manual means a compute node that executes a job.

The following sections describe each job classification and each type of job.

1.2.1 Job model

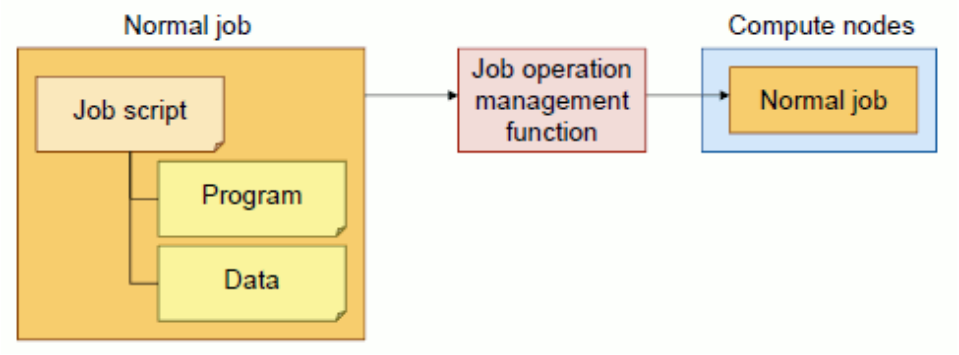
A classification by job structure is called a "job model." The following sections describe features of each job model.

1.2.1.1 Normal job

A normal job has the simplest structure.

One job script is executed for the job. The job ends when the job script ends.

Figure 1.2 Normal job



1.2.1.2 Bulk job

A bulk job consists of multiple instances of the same normal job submitted at the same time for execution.

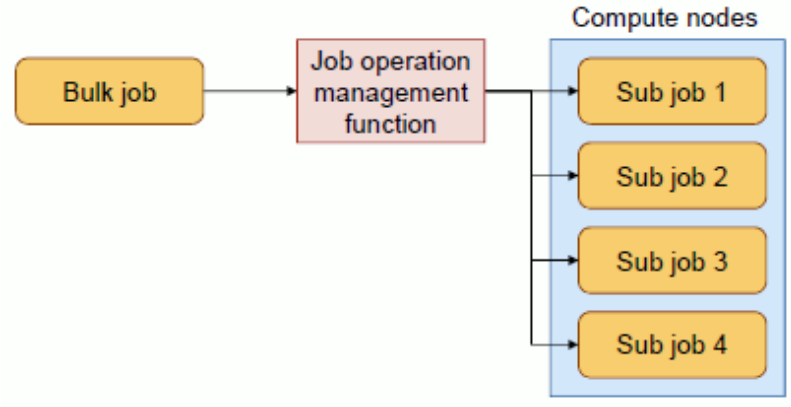
For example, suppose the user wants to change the job parameters and check the execution results for each change. The user would need to submit one normal job for each change. However, by using a bulk job, the user can submit multiple patterns at one time for one job.

The unit of processing of the job script executed at the same time in a bulk job is called a "sub job." Each sub job of a bulk job has a serial number (0 to 999999) set in the job. This number is called a "bulk number."

Note

Only a job that is batch and normal job can be a sub job of a bulk job.

Figure 1.3 Bulk job



1.2.1.3 Step job

A step job is a group of jobs that have an execution order or dependency relationship.

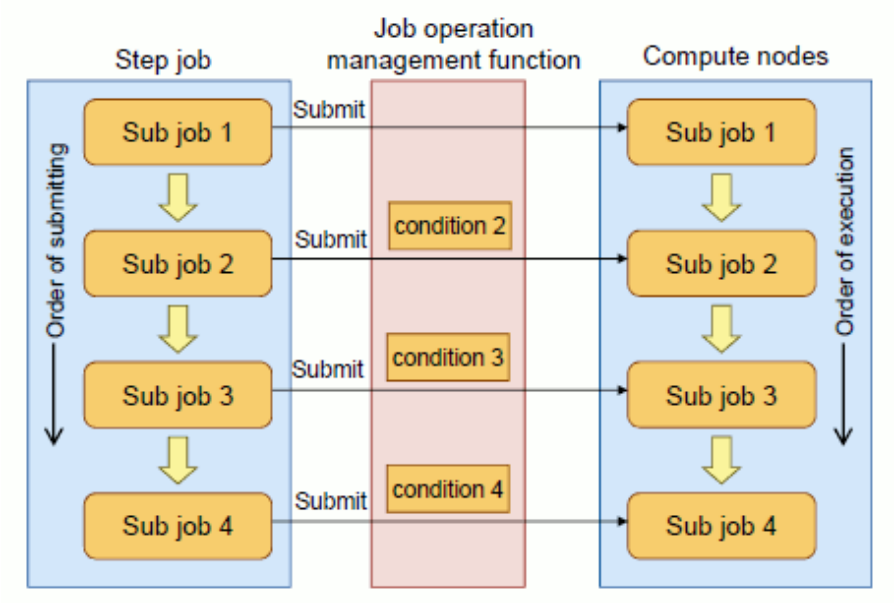
For example, suppose that whether a job is executed depends on the execution results of another job. If the job is a normal job, the user determines the job as such and submits the job. However, if the job is a step job, the user can specify the execution conditions and execution order for automatic processing according to the execution results of a specific job, when submitting the job.

Each job associated as a step job is called a "sub job." Each sub job of a step job has a serial number (0 to 65534) set in the job. This number is called a "step number."

Note

Only a job that is batch and normal job can be a sub job of a step job.

Figure 1.4 Step job

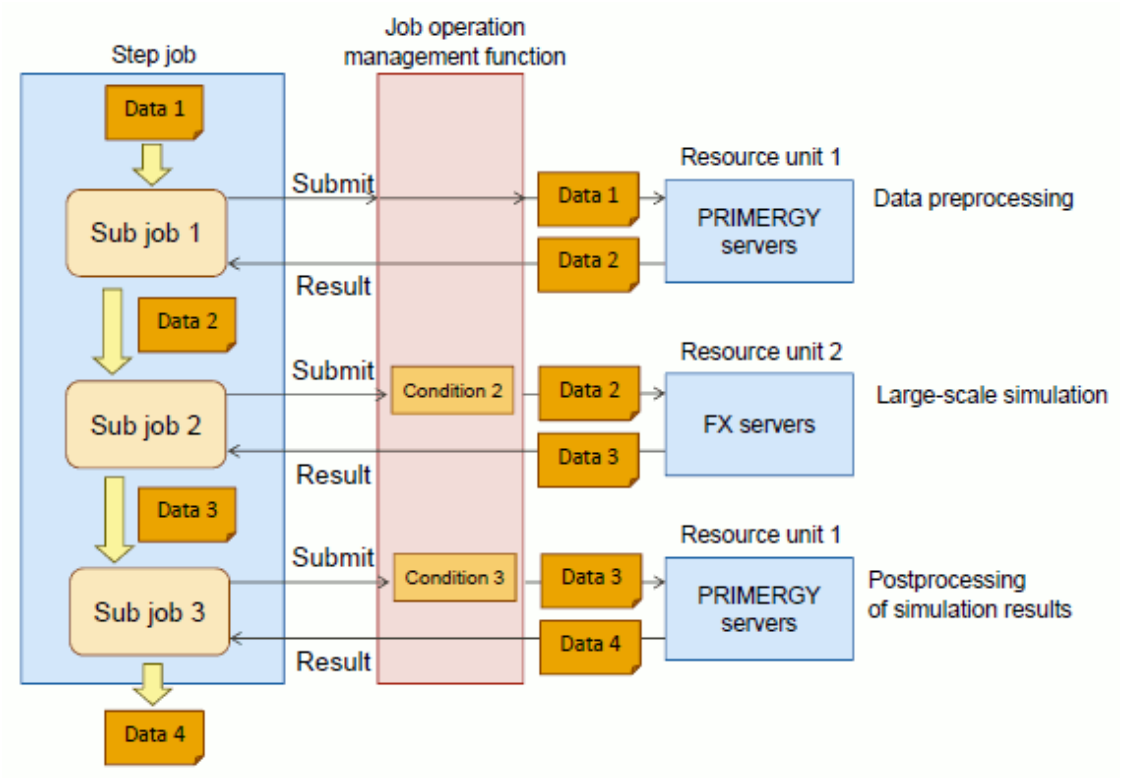


When submitting a step job, you can specify different resource units or groups for its sub jobs.
For example, suppose there is a cluster consisting of a resource unit of an FX server and a resource unit of a PRIMERGY server. You can submit applications, each of which is optimized for each architecture, as the sub jobs of a step job and execute them as a series of processes, as indicated below.

1. Sub job 1
   Preprocesses data for a large-scale simulation on a PRIMERGY server.
2. Sub job 2
   Takes the preprocessed data as its input and executes an application for a large-scale simulation on an FX server.

3. Sub job 3
   Takes the results of the large-scale simulation as its input and postprocesses the results (data analysis, visualization, etc.) on a PRIMERGY server.

Figure 1.5 Example of submitting individual sub jobs of a step job to different resource units



1.2.1.4 Workflow job

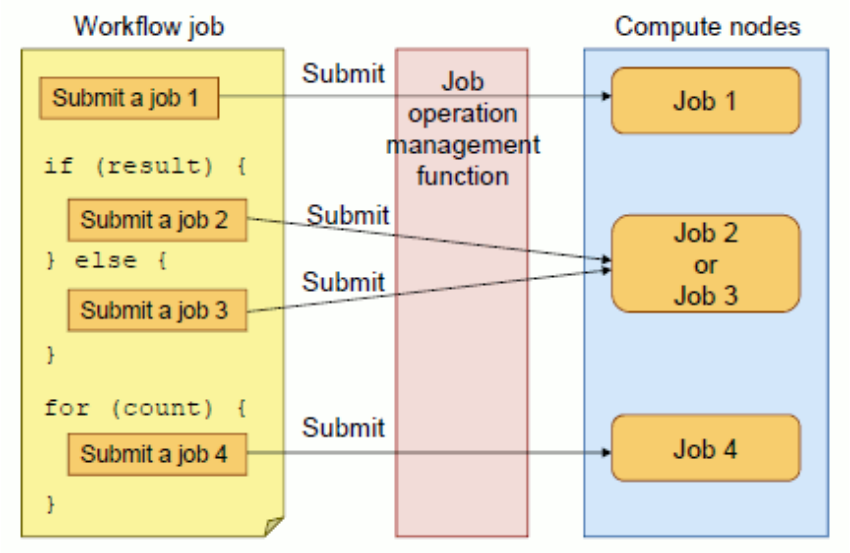
A workflow job is a group of jobs with the execution controlled (conditional branching and repetition) for each job.

Like with a step job, the next job to be executed depends on the execution results of another job. This execution control is more flexible than with a step job, depending on the contents of the shell script used to control execution.

Information

Specifically, a workflow job is not a type of job but a method using a shell script for user control over the submission of multiple jobs.

Figure 1.6 Workflow job



1.2.1.5 Master-Worker Job

Master-worker jobs are one of the job models, like normal jobs, step jobs, and bulk jobs, and they have the following characteristics.

- A master-worker job consists of a job script process, master process, and worker process. The master and worker processes perform computing tasks in collaboration with each other. (Task: Unit of parallel program processing)
- The master process controls entire computing tasks, generates and manages worker processes, and integrates computed results. Worker processes perform computing tasks upon request from the master process and then return the results to the master process.
- The master-worker job continues as long as the job script process is running, even if the compute node allocated to the master-worker job fails or the process ends abnormally.

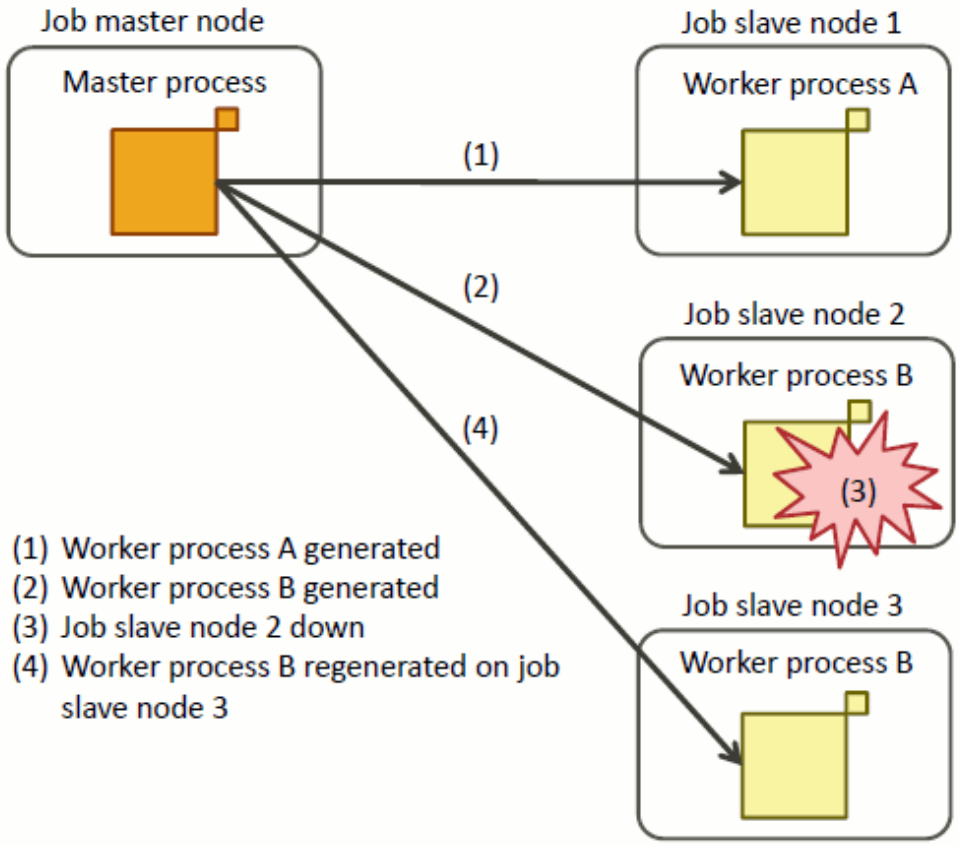
The user can use this characteristic to continue a computing task by creating a mechanism to re-execute a worker process on a different node when a compute node is down or the worker process has ended abnormally.

See

For details of the master-worker job, see "Job Operation Software End-user's Guide for Master-Worker Job."

The following diagram shows a conceptual image of a master-worker job executing a user program.

Figure 1.7 Conceptual image of execution of a master-worker job



Information

- The job script process runs on a node called a "job master node," and the other nodes are called "job slave nodes." The master process must run on the job master node because the purpose of the master process is to manage the worker processes running on job slave nodes.
- In terms of generating multiple processes, "bulk jobs" are a similar job model. In that method, a bulk job submits the same job script as multiple sub jobs, and each of these sub jobs runs independently. The bulk job, when submitted, specifies the number of sub jobs. The number does not change during execution.
  Meanwhile, in a master-worker job, the master process runs together with each worker process as one job while the processes communicate with one another. The number of worker processes changes during execution of the job since they are dynamically generated by the master process.

1.2.2 Batch job and interactive job

Jobs are classified into batch jobs and interactive jobs according to their execution format. They are called "job types."

Table 1.3 Batch job and interactive job

Type of job	Description
Batch job	A job which is executed un-interactively. The standard output and standard error of a job are output to files.
Interactive job	A job in which the interactive operation is possible for the program. The standard input/output of a job is the standard input/output of the shell performing the job execution request operation. Interactive jobs are mainly used to debug jobs and programs.

1.2.3 Sequential job and parallel job

Jobs are classified into sequential jobs and parallel jobs according to the program parallelism.

Table 1.4 Sequential job and parallel job

Type of job	Description
Sequential job	A sequential job is a job for executing a sequential process (single thread, single process). Example: Single-thread or single-process program
Parallel job	A parallel job is a job for executing a parallel processing program using multiple threads and/or multiple processes. Parallel processing with multiple processes may be executed within a single node or across multiple nodes. Example: Thread parallel processing program (automatic parallelization program, OpenMP program), process parallel processing program (MPI program), hybrid parallel processing program A job that executes an MPI program may be called an "MPI job."

Information

Program parallelization in the Job Operation Software is classified into the following types, which include thread parallel processing and process parallel processing.

Table 1.5 Types of program parallelization

Parallelization type	Description
Thread parallel processing	Thread parallel processing is a method to divide one process into multiple threads and execute them in parallel on multiple CPUs (or multiple cores on a multicore CPU). Since each thread shares the memory space of the process as it runs, programming is relatively easy. However, the upper limit on CPUs used is the number of CPUs in the node, so a significant performance improvement cannot be expected. Automatic parallelization by a compiler and OpenMP are examples of thread parallel processing. For details, see the manual provided with Development Studio.
Process parallel processing	Process parallel processing is a parallel processing method using multiple processes. The required data for parallel processing is transferred in communication between processes. Since this method can execute individual processes on different nodes, a significant performance improvement can be expected from an increase in the number of nodes. However, it requires advanced programming techniques. Each of multiple processes for one node may run on a single thread. This is called flat parallel processing. An MPI program that uses MPI (message passing interface) is an example of process parallel processing. For details on MPI, see the manual provided with Development Studio.
Hybrid parallel processing	Hybrid parallel processing is a method using both thread parallel processing and process parallel processing.

1.2.4 Single node job and multinode job

Jobs are classified into single node jobs and multinode jobs according to the required number of nodes.

Table 1.6 Single node job and multinode job

Type of job	Description
Single node job	A single node job is a job that requires only one node for execution. A sequential job is a single node job. An example for a parallel job is one that uses multiple processes or threads within a node.

Type of job	Description
Multinode job	A multinode job is a job that requires multiple nodes for execution. An example is a process parallel processing program that spans nodes or a hybrid parallel processing program.

Information

If more than one node is allocated to a single node job, the Job Operation Software processes the job as a multinode job.

The following table lists the specific expressions in this manual for classifications by job parallelism and classifications by the required number of nodes.

Table 1.7 Single node jobs and multinode jobs

Required number of nodes	Parallelization technique	Job name
1 node	Sequential job	Single node (sequential) job
	Thread parallel processing	Single node (thread parallel processing) job
	Process parallel processing	Single node (process parallel processing) job
	Hybrid parallel processing	Single node (hybrid parallel processing) job
Multiple nodes	Process parallel processing	Multinode (process parallel processing) job
	Hybrid parallel processing	Multinode (hybrid parallel processing) job

1.2.5 Classification by node resource allocation method

Jobs are classified into node allocated jobs and virtual node allocated jobs according to the unit for allocating node resources.

Table 1.8 Node allocated job and virtual node allocated job

Types of job	Description
Node allocated job	Node resources are allocated in units of physical nodes to jobs. Jobs of this type are also called "node-exclusive jobs" because only 1 job exists per node. However, the job cannot always use all the resources in the node, depending on the resource allocation method and the setting of the upper limit value.
Virtual node allocated job	Node resources are allocated in units of virtual nodes to jobs. A virtual node is a set of a CPU core and memory in a node. A virtual node is a node resource used exclusively by the job to which it is allocated. A feature of this method is that node resources can be used more efficiently than with node allocation. A node can have multiple virtual nodes, so jobs of this type are called "node-sharing jobs" when they share 1 node. A job that is executed on a PRIMERGY server may be a node-exclusive or node-sharing job, depending on the virtual node allocation method. The FX server does not support the virtual node allocated job.

See

For details on nodes, virtual nodes, and node resource allocation methods, see "1.6 Node Resource Allocation."

1.2.6 Classification by I/O node allocation method [FX]

Jobs are classified into I/O-exclusive jobs and I/O-sharing jobs, depending on how the I/O nodes that handle the job's input and output are allocated, especially storage I/O nodes.

Table 1.9 I/O-exclusive job and I/O-sharing job

Types of job	Description
I/O-exclusive job	An I/O-exclusive job occupies a storage I/O node for its I/O processing.
I/O-sharing job	An I/O-sharing job shares a storage I/O node used for job I/O processing with other jobs.

See

For details about I/O-exclusive job and I/O-sharing job, see "[Allocating I/O nodes exclusively]" in "2.3.2.11 Allocating resources in consideration of I/O nodes [FX]."

1.3 Job States

The following table lists the changing states of jobs, from job execution request to job end.

Table 1.10 Job states

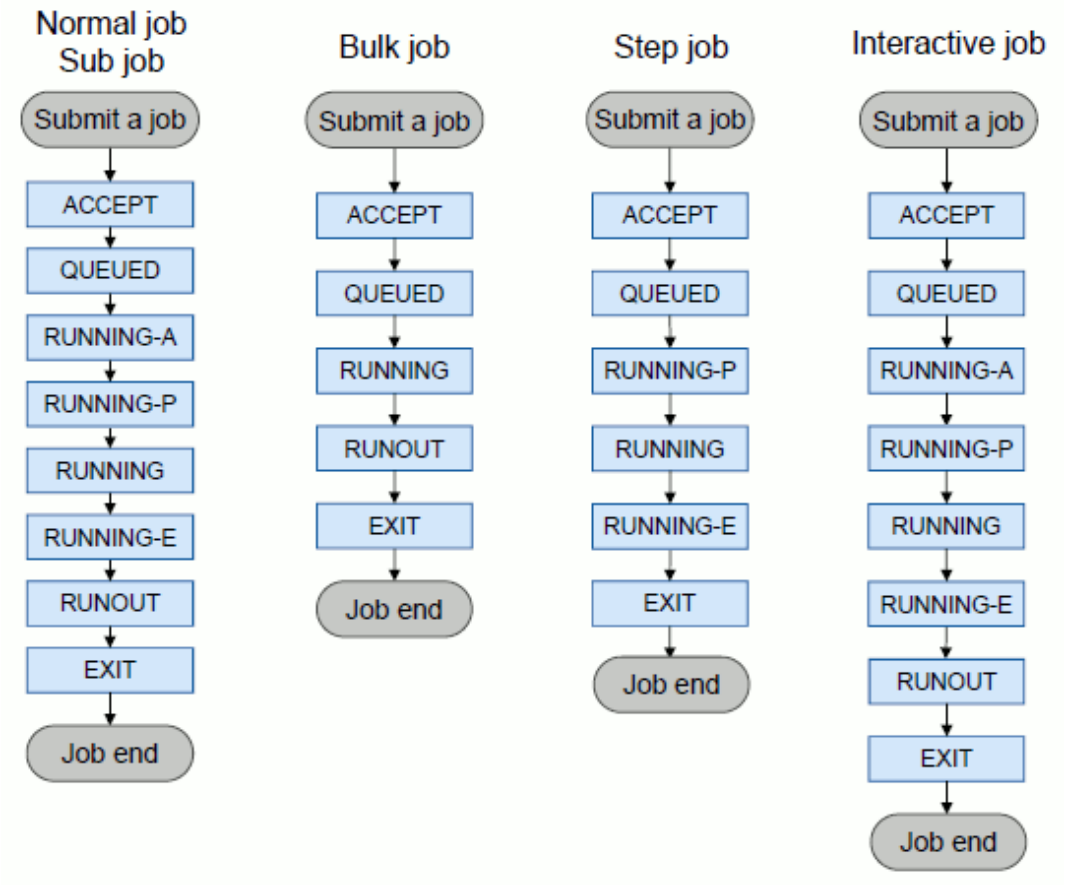
State name	Display (Note1)	Description
ACCEPT	ACC	Checking whether the job satisfies the acceptance conditions (items restricted with the job ACL function)
QUEUED	QUE	Waiting for a turn to execute the job, which has been accepted
RUNNING-A	RNA	Reserving the resources required for job execution
RUNNING-P	RNP	Executing the prologue process (Note 2)
RUNNING	RUN	Executing the job
RUNNING-E	RNE	Executing the epilogue process (Note 2)
RUNOUT	RNO	Job end processing is in progress
EXIT	EXT	Job end
REJECT	RJT	Job acceptance rejected
CANCEL	CCL	Job stopped by an instruction from the job submitter or the administrator
HOLD	HLD	Keeping the state of the submitted job from changing, when job execution has stopped
ERROR	ERR	Job stopped because of an error detected by the job operation management function
SUSPEND	SPP	During suspending process
SUSPENDED	SPD	Suspending process is completed
RESUME	RSM	During resuming processi

(Note1) The character strings show the states of jobs displayed by the pjsub command, pjstat command, etc.

(Note 2) For the detail of "prologue" and "epilogue", see "1.12 Prologue and Epilogue Function."

The basic state transitions of jobs depend on the type of job as shown below.

Figure 1.8 Basic state transitions of jobs



Note

- The bulk job and step job can have a state as one job and a state as a subjob. The "bulk job state" or "step job state" means the former. The bulk job and state job state transitions as well as their subjob state transitions differ as shown in the figure above.
- The step job and the running sub job have the same state (the next sub job, if there are no running sub jobs). However, if the sub job state is RUNNING-A, the step job state is QUEUED. Also, if the sub job state is RUNOUT, the step job state is RUNNING or RUNNING-E.

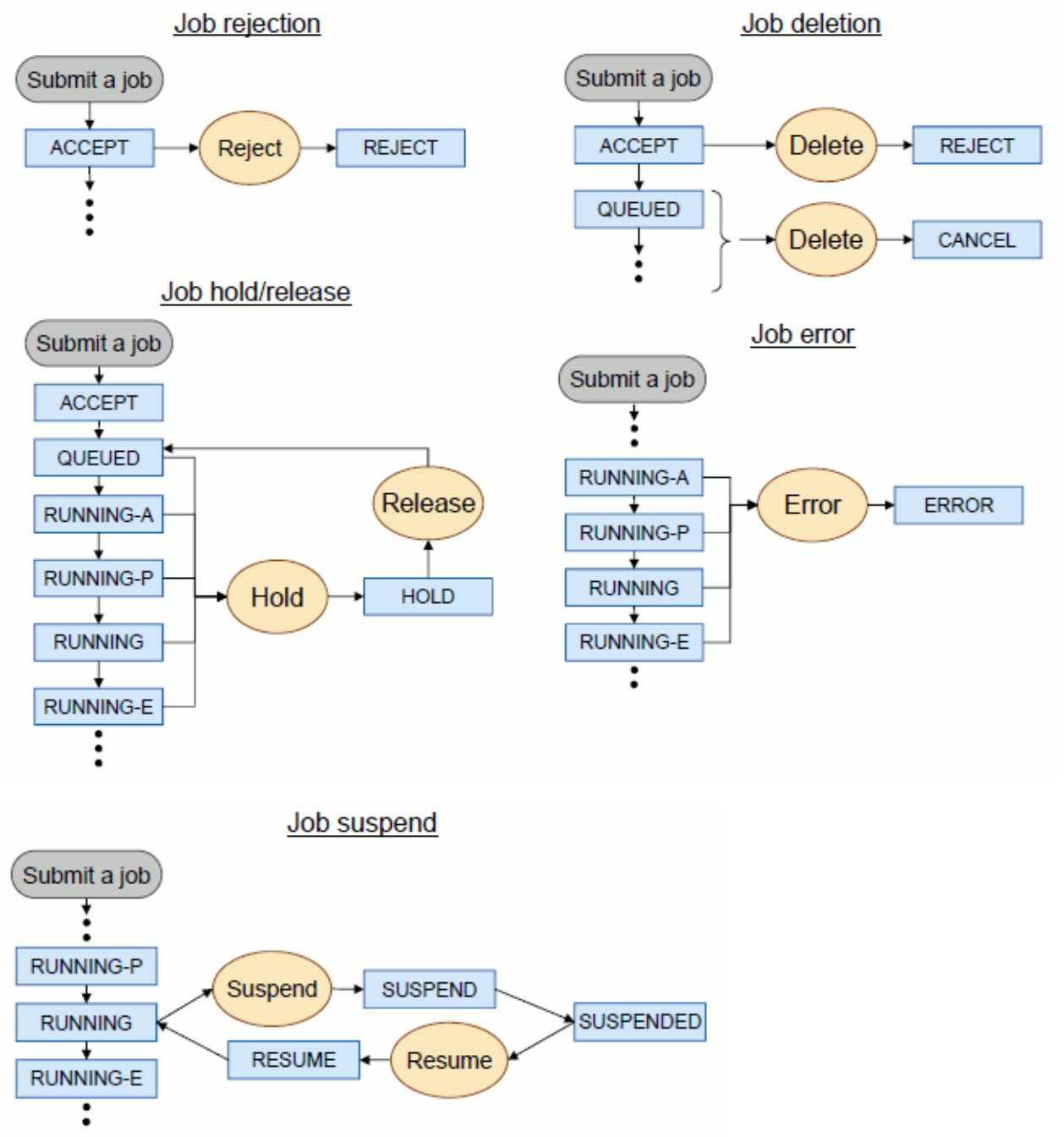
Job state transitions also depend on the events occurring after job submission, as described below.

Table 1.11 Events and job state transitions

Event	Job state transition
Job rejection	The job submitted by a user enters the ACCEPT state, indicating tentative acceptance. If the job does not satisfy the acceptance conditions (job ACL), it enters the REJECT state, indicating job rejection. For details on the job ACL, see "1.10 Job ACL Function."
Job deletion	The operation to cancel a submitted job is called "job deletion." When a job in the ACCEPT state is deleted, it enters the REJECT state. When a job in another state is deleted, it enters the CANCEL state. If the job is running, it is stopped. For details on the job deletion operation, see "2.5.1 Deleting a job."
Job hold and release	The operation to stop job execution but not change the job state is called "job hold." A job in the QUEUED, RUNNING-A, RUNNING-P, RUNNING, or RUNNING-E state can be held, entering the HOLD state as a result. Canceling the HOLD state restores the QUEUED state. For details on the job hold and release operations, see "2.5.3 Holding a job and canceling the hold."

Event	Job state transition
Job error	If an error occurs in the job operation management function for a job between the RUNNING-P and RUNNING-E states, the job enters the ERROR state.
Suspending a job	When a suspending occurs for a job in the RUNNING state, the job state becomes SUSPEND. Once the suspending process completes, the state becomes SUSPENDED. When the resuming process of a job that has been suspended (SUSPENDED) starts, the job state becomes RESUME. When the resuming process completes, the state becomes RUNNING, and execution of the job continues.

Figure 1.9 State transitions due to job rejection, error occurrence, etc.



1.4 Job ID and Sub Job ID

Each job is automatically assigned a "job ID," which is a unique identification number in the system. A job ID is a value ranging from 0 to 2147483647.

Among jobs, each bulk job and step job is also assigned a "sub job ID," which is a unique number identifying a sub job. A sub job ID is a character string consisting of a bulk number or step number added to a job ID. A bulk number is a value ranging from 0 to 999999, a step number is from 0 to 65534.

The sub job ID of a bulk job is written as "*job ID*[*bulk number*]". The sub job ID of a step job is written as "*job ID_step number*".

```
[Example]
Job ID                      : 123456
Sub job ID of bulk job   : 123456[1234]      Job ID 123456 and bulk number 1234 of sub job
Sub job ID of step job   : 123456_1234       Job ID 123456 and step number 1234 of sub job
```

1.5 Job Output

The standard output and standard error output of a job are output as separate files to the current directory at the job submission time.

For the file names, see "2.3.2.10 Specifying the standard output and standard error output files of a batch job."

1.6 Node Resource Allocation

To execute a job, the node resource (CPU cores and memory) must be allocated for executing the job.

Nodes in the system are grouped hierarchically in resource units. A resource unit is a unit for job operations in a cluster. A cluster is an operational unit of the system. Furthermore, a resource unit is divided into units for resource allocation, which are called resource groups. Multiple resource groups are prepared in accordance with the job operation policy. (For details, see the "Job Operation Software Overview" manual)

A job is executed in a resource group. If the administrator has not set the default resource group with the job ACL function, it is necessary to specify a resource group at the time of job submission. The job process monopolizes the allocated CPU core. CPU core binding can be controlled by using the Development Studio.

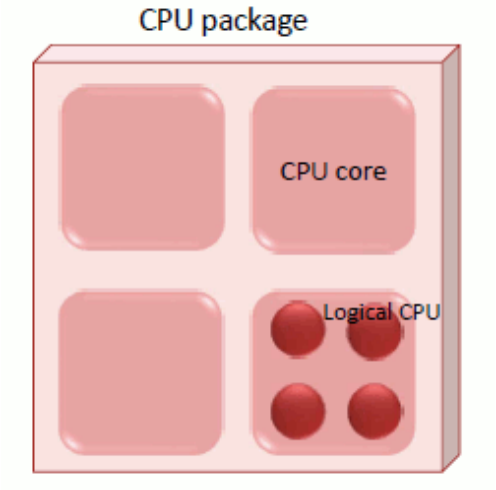
When submitting a job, the user specifies how to allocate the node resource in the resource group. Since the FX server and PRIMERGY server have different architectures, their concepts on how to allocate the node resource are also different.

The following sections describe the node resource allocation concepts for the respective compute nodes of the FX server and PRIMERGY server.

1.6.1 Hierarchical structure of CPU resources

The Job Operation Software handles CPU resources in the hierarchical structure shown below.

Figure 1.10 Hierarchical structure of CPU resources

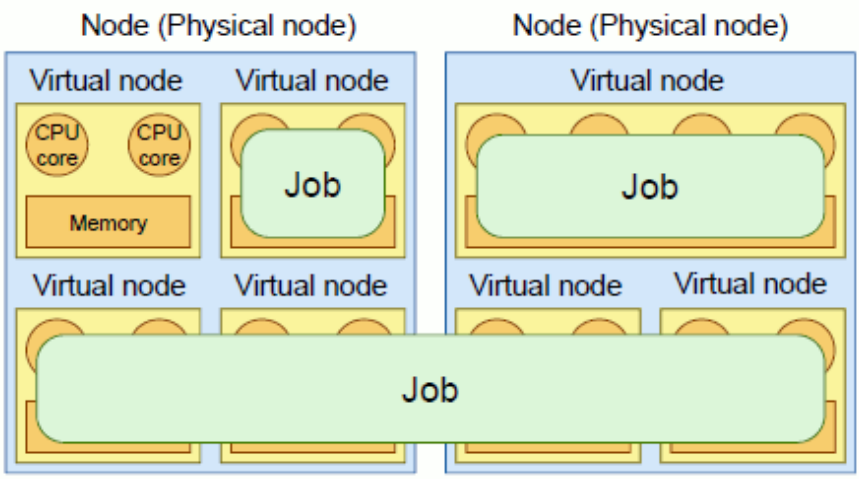


Name	Description
CPU package	Representation of the unit of a physical CPU
CPU core	CPU package has at least 1 CPU core. A CPU core has a CPU core ID that is set by the OS. A CPU core is the basic unit of CPU resources allocated to jobs by the Job Operation Software.
Logical CPU	Pseudo-CPU that exists in a CPU core when an SMT (*Simultaneous Multithreading*) function such as Intel(R) Hyper-Threading Technology is enabled.
CPU	Unit of CPU resources recognized by processes on the OS. It represents a logical CPU when an SMT function is enabled and a CPU core when the function is disabled. A CPU has a CPU ID that is set by the OS.

1.6.2 Nodes and virtual nodes

There are two types of nodes according to the concept of node resources managed by the Job Operation Software: physical nodes and virtual nodes, each of which is a set of CPU cores and memory.
The user defines virtual nodes by selecting a quantity of resources from the available physical nodes to satisfy their requirements. Use of virtual nodes helps prevent system node resources from being wasted, thus improving system throughput and utilization.

Figure 1.11 Virtual nodes



Information

- The amount of resources (the number of cores and amount of memory) per virtual node must be within the range of one physical node. You cannot combine multiple physical nodes into one virtual node.
- When used with the Job Operation Software, the term "node" by itself refers to a physical node.

For the FX server, you can select whether to allocate resources in units of physical nodes or virtual nodes. For a PRIMERGY server, resources are always allocated in units of virtual nodes.

1.6.3 Allocation in units of nodes

When allocating node resources in units of nodes, note that the concept differs depending on the model of the compute node. This section describes how to allocate FX server and PRIMERGY server.

See

For details on how to specify a node when submitting a job, see "2.3.2.2 Specifying a node resource."

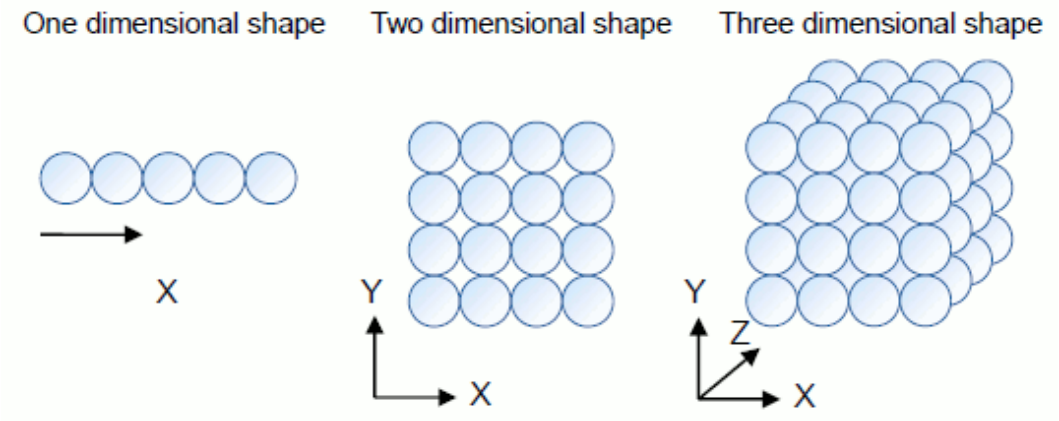
1.6.3.1 Allocation in units of nodes for FX servers

You can specify the following if the FX servers are allocated to the job in units of nodes.

- Node shape and number of nodes to allocate
- Method of placing nodes at Tofu coordinates
- Node allocation rules for executing an MPI program (Rank)

Specify the node to allocate as a shape placed in a one-dimensional, two-dimensional, or three-dimensional virtual space.

Figure 1.12 Node shapes (conceptual image)



There are three types of methods of placement according to how nodes of these shapes are placed at Tofu coordinates in the FX server: torus mode, mesh mode, and non-contiguous mode.

Table 1.12 Method of placing nodes at Tofu coordinates

Node placement method	Description
Torus mode	The minimum node allocation unit is one Tofu (12 nodes). Nodes selected for allocation are mutually adjacent in the Tofu coordinate space.
Mesh mode	The minimum node allocation unit is one node. Nodes selected for allocation are mutually adjacent in the Tofu coordinate space.
Non-contiguous mode	The minimum node allocation unit is one node. Nodes are selected for allocation so that they are mutually adjacent in the Tofu coordinate space as much

Node placement method	Description
	as possible. In the following cases, the selected nodes are not mutually adjacent: - There are no free nodes that are mutually adjacent. - Selecting nodes that are not mutually adjacent speeds up job execution start.

The characteristics of these placement methods are as follows:

- Job communication performance
  In terms of job communication performance, torus mode is advantageous.
  In either torus mode or mesh mode, communication of one job does not interfere with communication of another job. Note, however, that in mesh mode, communication performance may not always be constant because the lengths of inter-node communication paths may vary depending on the node placement result.
  In non-contiguous mode, communication of one job is likely to interfere with communication of another job because the allocated nodes are not always mutually adjacent.

- Ease of node allocation
  In terms of ease of node allocation, mesh mode and non-contiguous mode are advantageous because the minimum node allocation unit is one node.
  In other words, job execution may start earlier in mesh mode or non-contiguous mode than in torus mode.
  In addition, in mesh mode, the shape of the allocated nodes needs to fulfill certain requirements, although allocation is made in node units (see "Table 2.4 Size of nodes that can be allocated [FX]" in "2.2.1 Checking resource units and resource groups"). Therefore, node allocation is easier in non-contiguous mode than in mesh mode because the former mode places no restrictions on the node shape.

- Tolerance to communication path failure
  In terms of the tolerance to communication path failure, torus mode and mesh mode are advantageous.
  Non-contiguous mode tries to place nodes so that they are mutually adjacent as much as possible at Tofu coordinates, but they may not always be mutually adjacent. As a result, the Tofu interconnect on a node that is not allocated to a job may be used as a communication path for multiple jobs. Therefore, if the Tofu interconnect on the node used as a communication path becomes faulty, the multiple jobs are likely to be affected by this fault.

Note

- When a single node job with a one-dimensional shape is submitted in Torus mode, the allocated node is one node. In other words, a number of single node jobs with a one-dimensional shape can be executed in one Tofu node unit.
  Depending on the setting of the resource group to be executed by the administrator, a multinode job with up to 11 nodes allocated can undergo multiple executions in one Tofu unit.
  In Torus mode, when a single node job is submitted with a two-dimensional shape (1 x 1) or a three-dimensional shape (1 x 1 x 1), the allocated nodes are rounded up in Tofu units.

- If you specify nodes with a two-dimensional shape in mesh mode, the allocated node shape is rounded up to the minimum unit of 3 x 1.

- Suppose that a job in torus or mesh mode and a job in non-contiguous mode are mixed when executed within a resource unit. The job in non-contiguous mode may degrade the communication performance of the job in torus or mesh mode.
  Ask the administrator whether such jobs, including other users' jobs, can be mixed when executed within a resource unit.

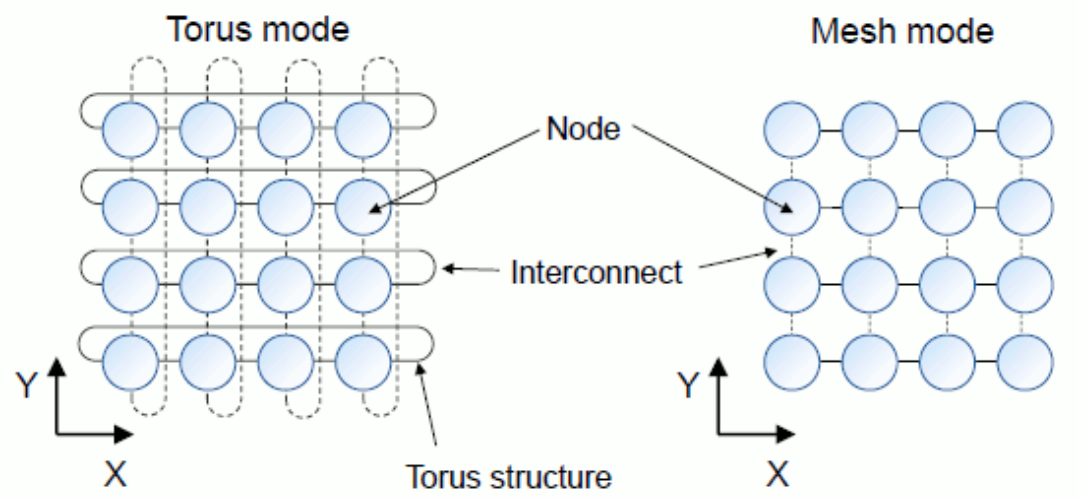
In torus mode, an interconnect that connects nodes consists of torus, each of which is along one axis. A torus is one of the shapes (topologies) for connecting nodes. It is a loop-shaped structure of nodes connected by interconnects. Each node at both ends is connected as the neighboring nodes along one axis of the node shape.

- Torus along the X-axis for a one-dimensional shape

- Torus along the X-axis and Y-axis for a two-dimensional shape

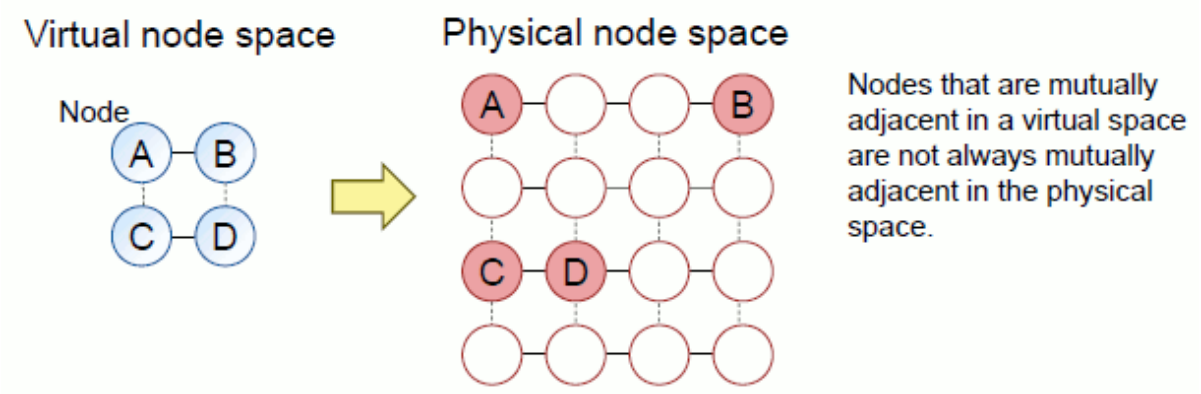
- Torus along the X-axis, Y-axis, and Z-axis for a three-dimensional shape

Figure 1.13 Connection between nodes (example of a two-dimensional shape)



In torus or mesh mode, nodes are allocated so that they will be mutually adjacent in the physical space. In contrast, in non-contiguous mode, allocated nodes are not guaranteed to be mutually adjacent in the physical space.
In other words, in non-contiguous mode, the inter-node distances in the virtual node space assumed for a job may not match the inter-node distances in the physical space.

Figure 1.14 Image of node allocation in non-contiguous mode



In torus mode and mesh mode, the maximum dimensions that can be specified for a node shape depend on the physical node configuration. The administrator specifies the maximum dimensions for each resource unit and resource group, which are units for job operations.
If the specified node shape exceeds the maximum dimensions, node resources become insufficient, and jobs are not executed. Users are requested to check the maximum dimensions for a node shape before submitting their jobs. For details on how to check it, see "2.2.1 Checking resource units and resource groups."

In torus mode and mesh mode, if a two-dimensional or three-dimensional node shape is specified and does not fit within the maximum dimensions as is, it may fit when rotated. If so, it is automatically adjusted in that way.
For example, suppose the maximum node dimensions are X x Y x Z = 2 x 3 x 32. You can specify a node shape of 6 x 3 x 2 even though it does not fit as specified. This is because the node shape fits within the maximum dimensions of 2 x 3 x 32 when rotated to 2 x 3 x 6.
In contrast, a node shape of 4 x 4 x 4 does not fit within the maximum dimensions of 2 x 3 x 32 no matter how it is rotated. For this shape, node resources would be insufficient, and the job would not be executed.

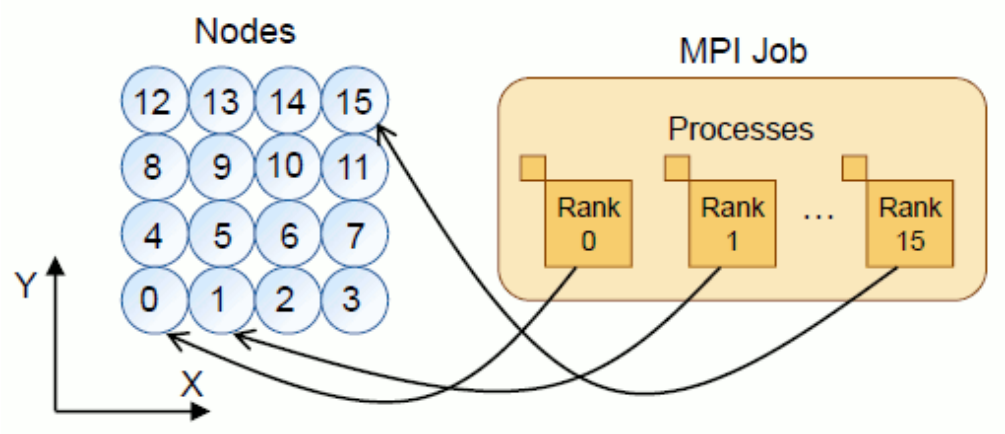
The automatic rotation of the node shape can be controlled. For details, see "2.3.2.2 Specifying a node resource."

It is necessary to also consider the following when determining the shape of a node group and the number of nodes.

- The node shape affects the cost of communication between nodes in a parallel program.
- An MPI program in the MPMD (multiple program multiple data) model handles the processing between multiple, different programs. It is necessary to consider the number of nodes required by each of these programs.

In MPI, a number called "rank" for process management is assigned to each process in the order of generation. You can specify which node is allocated to which rank in the Job Operation Software, when submitting a job.

Figure 1.15 Example of ranks and node allocation



Users allocate nodes by writing either a rule or an instruction for one-to-one correspondence between ranks and nodes. For details, see "2.3.6 Submitting an MPI job."

For MPI jobs in torus mode and mesh mode, it is better to use a two-dimensional or three-dimensional shape than a one-dimensional shape to assign ranks so that the nodes are adjacent to one another. The communication distance between ranks is shorter, and job performance can thus be expected to improve.

Note

- In torus mode and mesh mode, job execution with a specified two-dimensional or three-dimensional shape may start with later allocation of resources than with a specified one-dimensional shape.
- TCP/IP jobs should be run on no more than 512 nodes. At a larger scale, the system may be affected, such as abnormal termination of jobs due to a communication error.
  TCP/IP is generally slower than MPI using RDMA, so it is not suitable for use in jobs.

1.6.3.2 Allocation in units of nodes for PRIMERGY servers

To allocate PRIMERGY servers as node resources in units of nodes, specify the number of nodes.

Information

- Unlike FX servers, PRIMERGY servers do not have the concept of shape of nodes to be allocated.
- Administrators can set priorities for the nodes to be allocated (priority control for nodes allocated).

1.6.4 Allocation in units of virtual nodes

You can specify the following for node allocation to node-sharing jobs in units of virtual nodes:

- Number of virtual nodes to allocate
- Number of CPU cores and amount of memory per virtual node
- Amount of memory per CPU core
- Node resource allocation concept (node selection policy) [PG]

Note

The FX server does not support allocating virtual nodes.

When a PRIMERGY server is used, you can specify a node selection policy when submitting a job. The policy is a concept for node resource allocation.
The following table lists node selection policies.

Table 1.13 Node selection policy

Policy name		Description
Relationship between nodes and virtual nodes	Virtual node placement policy	Concept of placement of virtual nodes as opposed to nodes
	Rank map	Concept of setting virtual node IDs for the virtual nodes placed at nodes
Relationship between jobs and nodes	Node selection method	Concept of selecting the nodes used by jobs
	Priority control of allocated nodes	Concept of selecting nodes according to their own set priority
Relationship with other jobs	Execution mode policy	Concept related to occupation of nodes

This section describes these concepts.

Virtual node placement policy

A virtual node placement policy indicates how to allocate virtual nodes, for selected nodes.
The following table lists the policies.

Table 1.14 Types of virtual node placement policy

Method	Description
Absolutely PACK	All the virtual nodes are placed on one node. If they cannot be placed there, the job is not executed.
PACK	Virtual nodes are placed on as few nodes as possible.
Absolutely UNPACK	Only one virtual node is placed on one node. If there are more virtual nodes than nodes, the job is not executed.
UNPACK	If at all possible, only one virtual node is placed on one node. If there are more virtual nodes than nodes which are allocated to a job, multiple virtual nodes are placed on one node.

Rank map

A rank map describes rules on setting virtual node IDs for the virtual nodes placed in accordance with the virtual node placement policy. This corresponds to the specification of rank in MPI.
The following table lists the two methods for rank maps.

Table 1.15 Types of allocation methods with a rank map

Method	Description
rank-map-bynode	Virtual node IDs are set one by one according to the node ID order of a node (round robin method).
rank-map-bychip	Virtual node IDs are set sequentially starting with the virtual nodes within the same node.

Node selection method

The node selection method specifies whether to distribute the allocation of nodes to jobs or concentrate allocation to a few nodes. However, users cannot specify the node selection method. The contents set by the administrator with the job ACL function are applied.

For distribution of nodes that are to be selected, the nodes not being used (nodes with many available cores) have priority for selection. For concentration on a few nodes, a selection is made from the nodes already being used (nodes with fewer cores left available).

Priority control of allocated nodes

In the priority control of allocated nodes, the system selects nodes starting with that with higher priority according to the allocation priority set for each node by the administrator.

Execution mode policy

An execution mode policy is a method of specification taking into regard the occupation of nodes by jobs. The following table lists methods called execution mode policies.

Table 1.16 Types of execution mode policy

Method	Description
SIMPLEX	One job occupies a node. (Node-exclusive job) The node where the SIMPLEX specified job is running is not allocated to other jobs even if it has available CPU cores.
SHARE	A node is shared with other jobs that have the SHARE specification. (Node-sharing job)

See

- For details on how to specify a virtual node to allocate, see "2.3 Submitting a Job."
- For details on how to specify a node selection policy, see "2.3.5 Specifying a node selection policy [PG]."

1.6.5 NUMA allocation policy

For NUMA architecture compute nodes, the cost of accessing memory shared among multiple CPU cores is not uniform. Depending on the CPU core allocated to the job, this may cause variation in or deterioration of the execution performance. Therefore, the administrator can select the job allocation rules (NUMA allocation policy) for CPU core groups that have the same cost of accessing memory.

Table 1.17 NUMA allocation policy

Policy	Description
pack	The job is allocated so that it fits in one NUMA node as much as possible. (default)
unpack	The job is allocated so that it spans NUMA nodes.

End users cannot select the NUMA allocation policy. However, end users can confirm which policy is set for the system in use by checking the setting contents of the job ACL function (see "1.10 Job ACL Function").

Information

The NUMA allocation policy has meaning when a virtual node is allocated to a job.

1.7 Job Execution Order

Submitted jobs are evaluated with the various elements shown below to determine an execution order based on the results. The administrator decides the elements and criteria for determining this execution order. This set of elements and criteria is called the job selection policy.

Table 1.18 Elements that determine the job execution order

Element	Description
Privileged jobs	Re-executed jobs are assigned higher execution priority.
Submission time	Jobs are executed in the order in which they were submitted. This is called fcfs (first-come first-serve).
Number of requested nodes [FX]	A job is evaluated by the number of nodes requested by it.

Element	Description
Group priority and user priority	Jobs belonging to a group or user are evaluated by the job priority set for the group or user.
Resource group priority	A job is evaluated based on the priority assigned to its resource group in the resource unit.
Priority of jobs of the same user	Multiple jobs of the same user are evaluated based on the priority assigned to them.
Job priority in a resource unit	A job is evaluated based on its priority in the resource unit.
Priority based on job execution results	Jobs are evaluated based on the past job execution results so that users can use the system on an equal basis. This is called the "fair share function."
Elapsed time limit value	A job is evaluated by its elapsed time limit value.
Elapsed time limit value x number of requested nodes [FX]	A job is evaluated by the product of its elapsed time limit value and the number of nodes requested by it.
Specification of execution start time	A job is evaluated based on whether its job execution start time is specified at the time of job submission.
Specification as an interactive job	A job is evaluated based on whether it is submitted as an interactive job.

Note

The job execution order is determined based on the above elements. However, if there are free resources available, some jobs may be executed earlier than in the normal execution order. This function is called the "backfill function." The administrator sets whether this function is to be enabled or disabled.

The following sections describe details of the elements which determine the job priorities.

1.7.1 Group, user, and resource group priorities

In some cases, job execution priorities are assigned to groups or users in the OS. These priorities can only be configured by the administrator using the job ACL function (see "1.10 Job ACL Function").
You can check priorities assigned to groups or users by using the pjacl command (see "2.2.2 Checking restriction information").

Furthermore, job execution priorities may also be assigned to resource groups in the Job Operation Software.
These priorities can only be configured by the administrator as well. To obtain information on the priorities assigned to resource groups, contact the administrator.

1.7.2 Job priorities

Users can set the execution priorities of only their own jobs. The top priority is 255, and the lowest priority is 0.

You can specify the job priority at the job submission time by -p option of the pjsub command. After a job is submitted, you can change the job priority by the -p option of the pjalter command.

See

For details on how to specify the priority at the job submission time, see "2.3.2.9 Specifying a job priority."

Each job is assigned two types of priorities: user-specific priorities and priorities applicable to all jobs within the resource unit. Priorities of the latter type can only be configured and changed by the administrator.

1.7.3 Fair share function

The fair share function determines execution priorities according to the results of computer resource usage for executed jobs, so that users and groups each have fair use of the system.
If the fair share function is enabled, the priority of a job decreases immediately after it is executed. The priority is restored at a constant rate as time elapses.
For this reason, if a large-scale job is executed or jobs are executed frequently, the subsequent jobs may not be immediately executed.
To find out whether the fair share function is enabled on your system, ask the administrator.

1.7.4 Job execution start time

Users can specify the job execution start time when submitting a job. To specify the job execution start time, use the --at option of the pjsub command when submitting a job. However, an interactive job cannot have a specified execution start time.

Note

Job execution does not necessarily start at the specified time. It depends on the availability of computer resources and job selection policy. If execution cannot start at the specified time, it starts afterward as soon as computer resources can be reserved.

See

For details on how to specify the job execution start time, see "2.3.2.8 Specifying an execution start time."

1.8 Custom Resource

The Job Operation Software allows you to define any resource, such as a software license and power. This arbitrary resource is called "custom resource." For example, suppose you define a software license as a custom resource and request the necessary quantity of software licenses when submitting a job. Then, the job is scheduled to be executed at a time when there is a sufficient quantity of software licenses for use in the job. By using custom resources in this way, you can schedule jobs according to various purposes with not only a hardware resource, such as the CPU or memory of a compute node, but also a general resource.

Only the administrator can define custom resources.

To specify a requested custom resource when submitting a job, use the -L option of the pjsub command.

Custom resources are defined as the two types described below. According to these types, specify the requested custom resource when submitting a job.

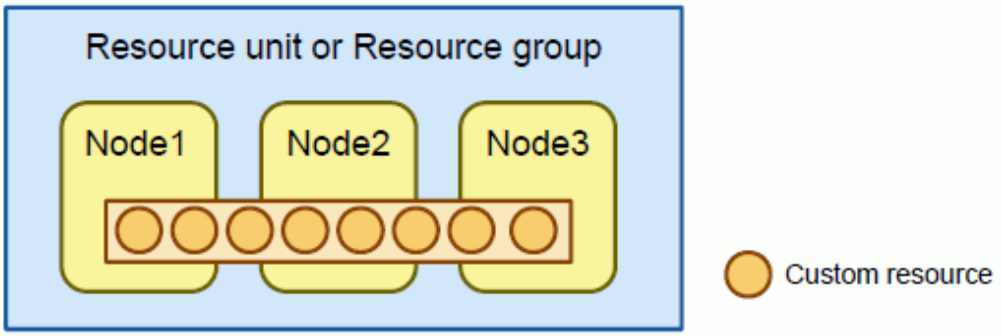
- Custom resources defined with a "numerical value"
  When submitting a job, specify the numerical value of resource that you want used in the job (requested quantity).
  For example, if the custom resource is defined with a numerical value and if the defined numerical number of this resource is eight, specify a numerical number that does not exceed eight.

- Custom resources defined with a "type"
  When submitting a job, specify the type that you want used in the job (requested type) from the types of multiple resources defined as custom resources.
  For example, if the custom resource is defined with a type and if both types a and b define this resource, specify either type a or b.

In addition to the above-described two types of custom resources, custom resources are defined in the two units described below. When submitting a job, you need to consider which unit to use in defining a custom resource and also specify the requested quantity or requested type of the resource.

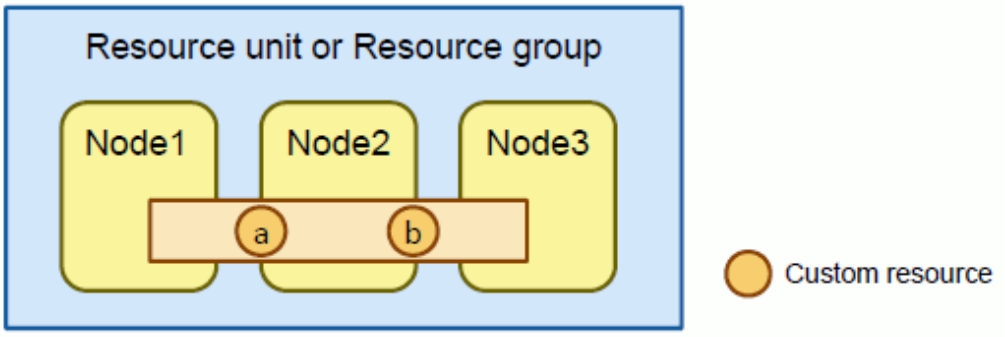
- Custom resources per resource unit or resource group
  For example, suppose the custom resource is defined with a numerical value. If the set number of this resource is eight, you can use eight of the resource in all the compute nodes inside a resource unit or resource group.

Figure 1.16 Custom resources per resource unit or resource group (when defined with a numerical value)



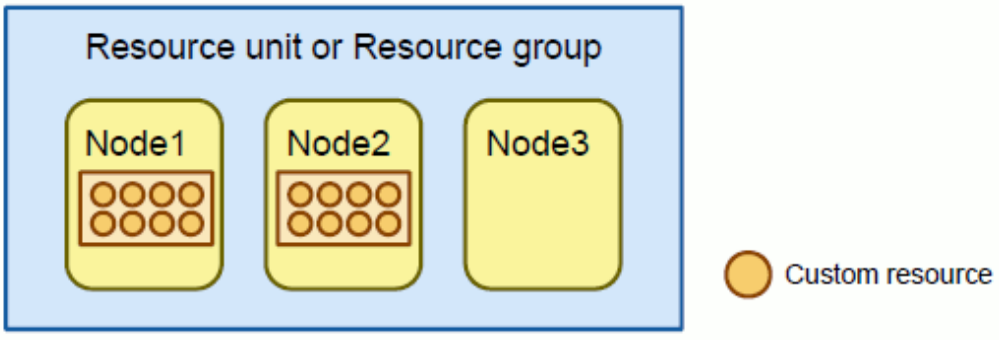
Suppose the custom resource is defined with a type. If both types a and b define this resource, you can use these two types as job resources in every compute node inside a resource unit or resource group.

Figure 1.17 Custom resources per resource unit or resource group (when defined with a type)



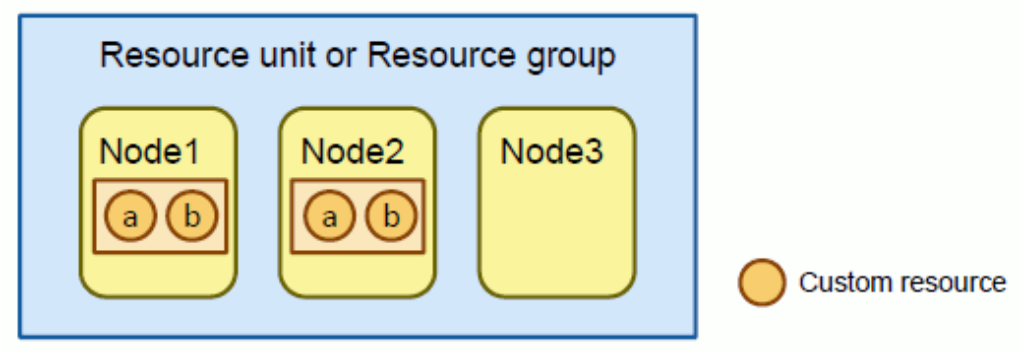
- Custom resources per node
  For example, suppose the custom resource is defined with a numerical value. If the set number of this resource is eight for the specified compute nodes (node 1 and node 2) inside a resource unit or resource group, you can use eight resources in each of the specified nodes.

Figure 1.18 Custom resources per node (when defined with a numerical value)



For example, suppose both types a and b define the custom resource. If types a and b are set for node 1 and node 2, you can use types a and b as job resources in each of the specified nodes.

Figure 1.19 Custom resources per node (when defined with a type)



See

For details on how to submit a job specifying a custom resource, see "2.3.2.4 Specifying a custom resource." For details on how to check a set custom resource, see "2.2.2 Checking restriction information."

1.9 Job Execution Environment

The Job Operation Software provides a function to switch explicitly the software environment (job execution environment) for executing job programs according to the user's specifications. The function is called the "job execution environment-customizing function." With this function, users can select the appropriate environment from the job execution environments deployed on the system to execute their own jobs in an environment tailored to the jobs. Furthermore, users can use execution environments that they prepared themselves.

The job execution environment-customizing function makes available the following environments (execution modes) as job execution environments:

Normal mode

The default job execution environment mode. Job programs are executed in the standard OS environment (Linux) installed on the system. This mode does not require explicit specification of the job execution environment.

Docker mode

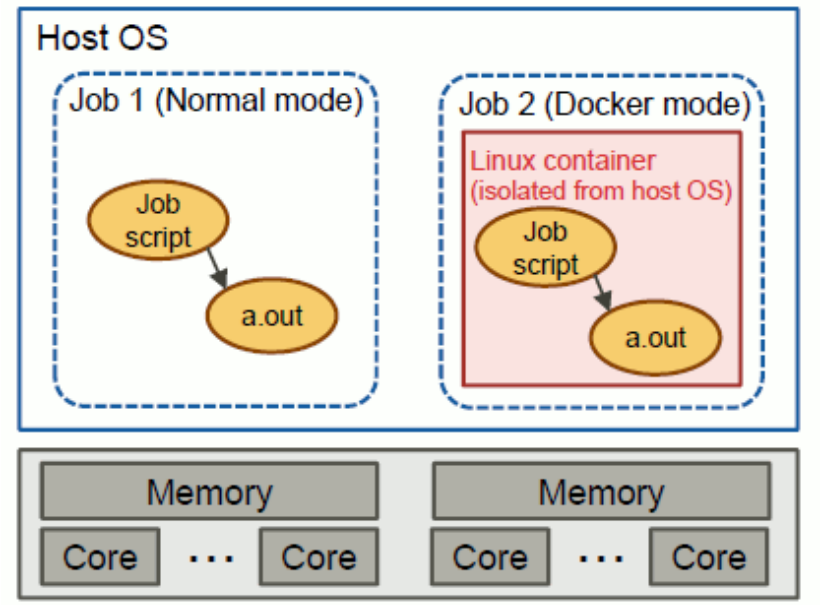
Docker is open-source software providing OS-level virtual environments using Linux containers. By preparing an environment, called a Linux container, isolated from the host OS, you can run a program on a different OS (container image) than the host OS.

See

For details on Docker, see https://www.docker.com/.

This mode is available on FX servers and PRIMERGY servers.

Figure 1.20 Docker mode



Docker mode executes all job programs, including users' job scripts, on the container started from the container image specified at the job submission time. Using docker mode, you can execute jobs by using the OS packaged as a docker image as it is. This makes it easy to introduce the necessary software environment for executing applications.

In docker mode, users can use not only the job execution environments that the administrator deployed on the system but also the job execution environments that they prepared themselves. The names for these two methods of use are as follows:

- SDI (System Deployed Image) specification

  In this method, select and use a job execution environment deployed in the system.

- UDI (User Defined Image) specification

  In this method, use a job execution environment prepared by the user and the user can switch the specified container image and submit the job.

Note

The availability of UDI specification depends on the system settings. To use UDI specification, contact the system administrator about the specification method.

McKernel mode

McKernel is a type of LightWeight Kernel (LWK), which is a lightweight OS operating in linkage with Linux. McKernel is published as open source software. McKernel mode executes users' job programs on McKernel. The job programs run only on McKernel, which means that the programs are executed in an environment isolated from the host OS. This can be expected to improve performance relative to applications, which are especially sensitive to system noise.

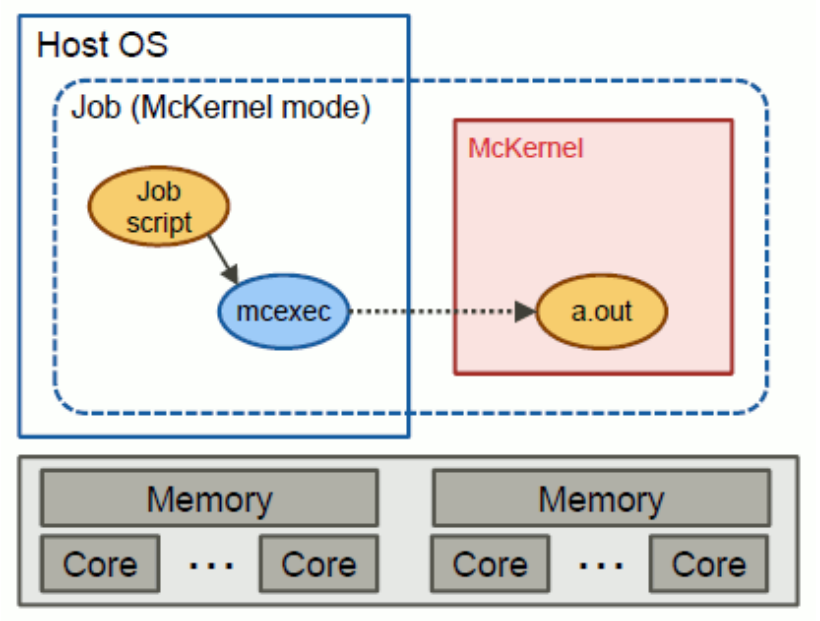
See

For details on McKernel, see https://ihkmckernel.readthedocs.io.

This mode is available only on FX servers.

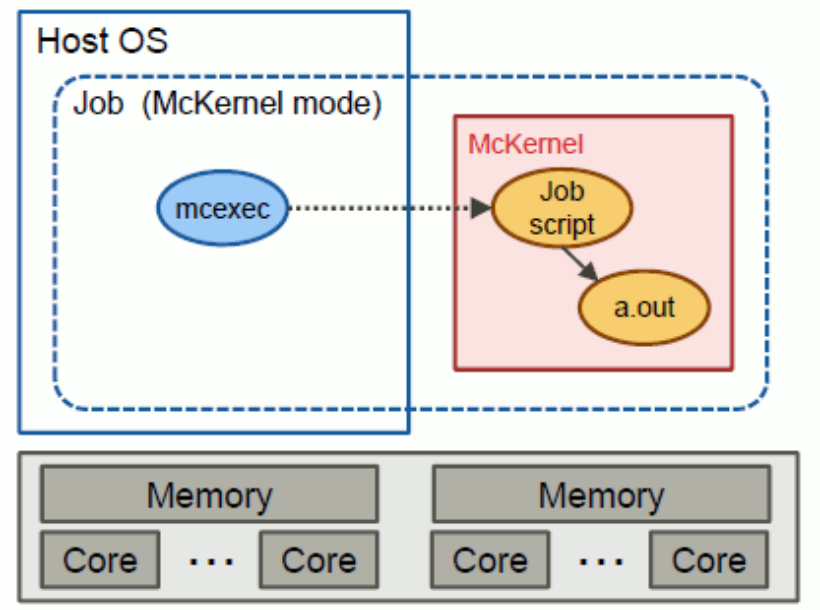
The following figure shows an outline of McKernel mode. McKernel aims to reduce the weight of the OS by transferring heavy OS processing, such as system calls, to the host OS to bring about the performance improvements of programs. The process in charge of this transfer processing is called mcexec. mcexec is started on the host OS.

Figure 1.21 McKernel mode (1)



By default, McKernel mode executes job programs on McKernel and starts job scripts on the host OS. However, if necessary, you can move all user processes, including job scripts, onto McKernel to also start them there according to the specifications at the job submission time. For details on the specification method, see "2.3.8 Specifying a job execution environment."

Figure 1.22 McKernel mode (2)



McKernel mode allocates resources to a job as described below.

- CPU
  - McKernel mode uses all the CPU cores in a compute node.
  - The programs started on the host OS and the mcexec command run on assistant cores.
- Memory

  The amount of memory specified at the job submission time is divided into amounts allocated to McKernel and programs running on the host OS.

- Out of the amount of memory specified in an option at the job submission time, 128 MiB is allocated to programs started on the host OS. The remaining memory is allocated to McKernel.
- The environment variable PJM_JOBENV_MCKERNEL_JOBMEM can change the memory amount allocated to programs started on the host OS. For details on how to set the variable, see "2.3.8 Specifying a job execution environment." If the environment variable PJM_JOBENV_MCKERNEL_JOBMEM has no setting, the administrator's set default value applies.

In McKernel mode, UDI specification and SDI specification are available. See the description of Docker mode for more information on UDI and SDI specifications.

KVM mode

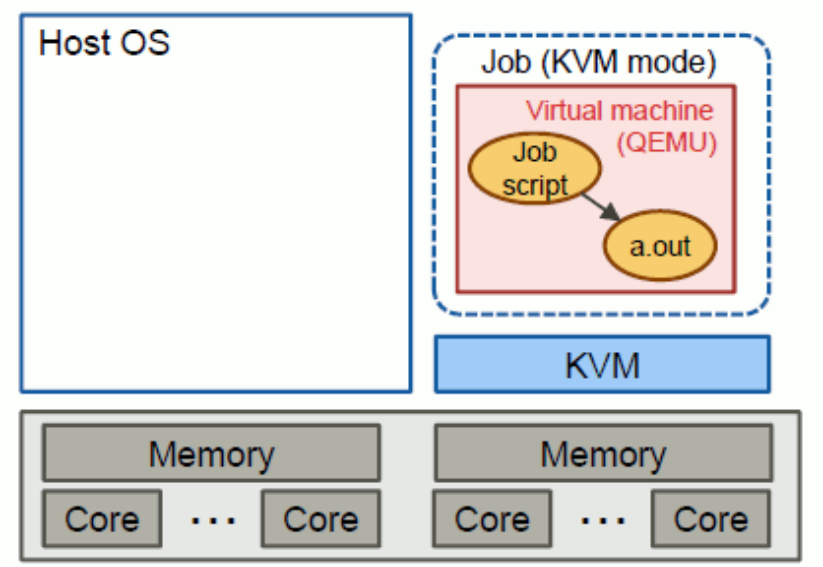
KVM (Kernel Virtual Machine) is a para-virtualization function on Linux. KVM mode executes users' job programs on this KVM. KVM mode uses KVM through the libvirt and the QEMU open source software for managing virtual machines. The hardware virtualization support enables job execution on a high-speed hardware virtualization environment. This is effective for users, including the system developers of OS kernel layers such as kernel modules, who require the privilege to execute programs.

See

For details on KVM, see https://www.linux-kvm.org/page/Main_Page.

This mode is available on FX servers and PRIMERGY servers.

Figure 1.23 KVM mode



In KVM mode, UDI specification and SDI specification are available. See the description of Docker mode for more information on UDI and SDI specifications.

Note

The KVM mode can be used with the system where user's home directory is shared by the login node and the compute node. The availability of UDI specification depends on the system settings. To use UDI specification, contact the system administrator about the specification method.

See

For details on how to submit a job specifying a job execution environment, see "2.3.8 Specifying a job execution environment."

To prepare a job execution environment on your own, you need to have knowledge about virtualization technology such as Linux containers. This manual describes only how to create image files for job execution environments in "E.1 Creating an Image File for a Job Execution Environment."

1.10 Job ACL Function

The Job Operation Software provides the "job ACL (Access Control List) function" to set upper limits on the memory and CPU time used by a job. The job ACL function also limits the functions available to users in accordance with the job operation policy.

The job ACL function has settings available for each cluster, resource unit, resource group, group, and user. The administrator configures settings appropriate to the job operation. The user can use the system within the restrictions by the job ACL function. For example, any jobs requiring resources or functions that exceed a restriction by the ACL function are rejected. Any job that reaches the job execution time limit is forcibly terminated.

You can use the pjacl command to check the details on the items restricted with the job ACL function.

See

For details on how to use the pjacl command to check the restrictions of the job ACL function and the restricted items, see "2.2.2 Checking restriction information."

1.11 Job Statistical Information

Information such as the amount of resources used by a job and various limit values is called "job statistical information." The Job Operation Software provides a function to output this job statistical information. You can analyze the job execution situation after the fact by referencing job statistical information.

Users obtain job statistical information as job output results by using the pjsub command when submitting a job. You can also check the statistical information on a running job by using the pjstat command.

See

For details on how to check job statistical information by using the pjstat command, see "2.4.2 Displaying job statistical information." For details on how to output job statistical information by using the pjsub command, see "2.3.2.6 Specifying job statistical information output."

For details on the items included in job statistical information and their meanings, see "Appendix A Job Information" and man manual pjstatsinfo(7).

1.12 Prologue and Epilogue Function

The administrator can set a specific script to be executed immediately before the start of and/or immediately after the execution of each job in accordance with the job operation policy. An example is a common process for all jobs, such as a process to set a specific environment variable or to prepare a work directory before job execution and delete it at job end.

This is called the "prologue and epilogue function."

Though users cannot change the prologue and epilogue function processes, keep the following in mind since they are executed as a part of jobs.

- The standard output and standard error output of the prologue and epilogue processes are output as job execution results.
- The resource limit values for jobs also apply to the prologue and epilogue processes.
- The contents added by the prologue and epilogue processes become job statistical information.
  However, whether the elapsed time of job execution includes the prologue and epilogue processes depends on the system settings. For details, contact the administrator.

For details on the prologue and epilogue processes in your system, contact the administrator.

1.13 Layered Storage System

The Job Operation Software supports job execution in the "layered storage system." The layered storage system is a file system with a hierarchical structure consisting of first-layer storage and second-layer storage, as described below.

Note

The layered storage system is currently available on systems with the FX servers.

- First-layer storage

  First-layer storage is a high-speed file system using the Lightweight Layered IO-Accelerator (LLIO) technology. First-layer storage is also called LLIO. In the layered storage system, first-layer storage is directly accessible from compute nodes.

  First-layer storage has the following three types of areas:

  - Node temporary area

    Each of the compute nodes allocated to a job can use this local area.

  - Shared temporary area

    This area can be shared between the nodes allocated to a job. Processes of the same job are also accessible from any compute node.

  - Second-layer storage cache

    From a job, this cache looks like second-layer storage. However, internally, the second-layer storage cache is located on first-layer storage. This cache is accessed instead of second-layer storage being accessed directly.

  The node temporary area and the shared temporary area are high-speed work file systems secured for each unit of execution shown in "Table 1.19 Units of execution for which the node temporary area and the shared temporary area are secured." They are available only in each unit of execution. These areas are secured when the unit of execution starts and deleted when the unit of execution ends.

Table 1.19 Units of execution for which the node temporary area and the shared temporary area are secured

Types of job		Unit of execution
Batch job	Normal job	Each job
	Step job	Each subjob
	Bulk job	Each subjob
	Master-Worker job	Each job
Interactive job		Each job

See

For details on first-layer storage, see "LLIO User's Guide."

- Second-layer storage

  Second-layer storage uses the FEFS distributed file system. The login node and each compute node share this storage.
  Before submitting a job, the user places job scripts and other files necessary for job execution, on this file system.
  The access from the job on a compute node to second-layer storage is internal access to cache on first-layer storage.

See

For details on FEFS, see "FEFS User's Guide."

1.14 Command API

The user interfaces required for job operations vary depending on the operating system. The job operation management function provides an interface for function calls from programs in order to support the creation of commands that have the user interfaces preferred by users. The called functions (job operation and information acquisition) are equivalent to commands provided by the management function. This interface is called the command API (Application Programming Interface).

The command API provides the following functions.

Table 1.20 Functions provided by the command API

Target user	Classification	API type
End-user and administrator	Job operation API	Submitting a job (corresponding to the pjsub command)
		Deleting a job (corresponding to the pjdel command)
		Holding a job (corresponding to the pjhold command)
		Releasing a job hold (corresponding to the pjrls command)
		Sending a signal to a job (corresponding to the pjsig command)
		Waiting for a job to end (corresponding to the pjwait command)
		Changing a job parameter (corresponding to the pjalter command)
	Information acquisition API	Acquiring job information (corresponding to the pjstat command and the -s/-S option of the pjstat command)
		Acquiring resource information (corresponding to the --rsc option of the pjstat command)
		Acquiring limit value information (corresponding to the --limit option of the pjstat command)
		Acquiring job ACL information (corresponding to the pjacl command)
		Acquiring the resource status (corresponding to the pjshowrsc command)
Administrator	System operation API	Changing whether a job is submitted and executed (corresponding to the --set-rsc-ug and --show-rsc-ug options of the pmpjmopt command)
	Job operation API	Changing a job parameter (corresponding to the pmalter command)

For details on the command API, see "Job Operation Software API user's Guide for Command API."

Chapter 2 Job Operation Procedures

This chapter describes the operations from creating a job to checking execution results.

Users create jobs, submit jobs, and check the execution results in the following workflow:

1. Creating a job
2. Confirming job control information
3. Submitting the job
4. Displaying job information
5. Deleting the job (Abort the job)
6. Confirming job results

Note

Unless otherwise noted, the operation examples in the following sections are performed on the login node.
Administrators can perform these operations not only on the login node but also with commands executed on other nodes. For details, see the man page of each command.

2.1 How to Create a Job

This section describes procedures related to creating and placing a job in an appropriate location.

2.1.1 How to create a job

A job script is, in essence, a shell script.

The following example shows the descriptions in a job script.

```
#!/bin/bash                                          (1)
#PJM -L "node=2"                                     (2)
# comment                                            (3)
# comment
export PATH=<dirname>:$PATH                          (4)
#PJM -L "elapse=86400"                               (5)

./a.out                                              (6)
```

(1) bash executes the job script.
(2) The fields delimited by a space after #PJM are handled like pjsub command arguments.
(3) Any line not beginning with #PJM is simply judged to be a shell comment.
(4) Environment variable settings, etc.
(5) Once a line other than a comment line appears, the subsequent lines are simply judged to be comment lines.
(6) The command executes the program a.out.

Note the following about writing job scripts.

- If no shell is specified at "#!" on the first line of the job script, the shell for executing the job script is the user's login shell.
- You can write the arguments of the pjsub command for submitting a job, on a line beginning with "#PJM " in a job script. The arguments of the pjsub command has priority over that specified in a job script.
- Once a line other than a comment line appears in a job script, "#PJM" on the subsequent lines is ignored with the lines simply treated as comments.
- For a job script, the user who submits the job requires read authority. Execution authority is not required.

- Do not redirect to /dev/stdout or /dev/stderr in a job script. If the script redirects to either, the standard output file or standard error output file is overwritten from the beginning.

The following sections describe points for users to take into consideration when creating a job script.

Information

You can write the options of the pjsub command for submitting a job, as described in "2.3 Submitting a Job," as a shell comment line in a shell script. For this reason, the following sections also describe the options of the pjsub command.

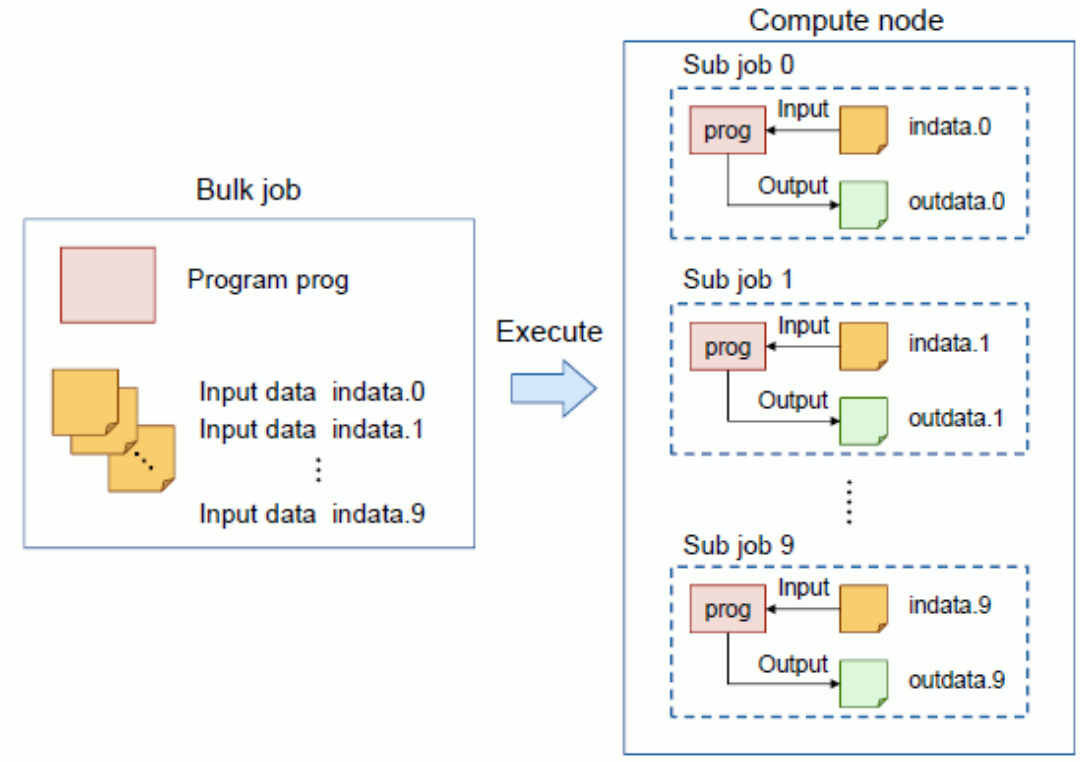
2.1.1.1 Creating a bulk job

A job script for a bulk job is designed such that input parameters of the job can be changed for each sub job.

For this reason, the bulk job uses the bulk number that is set for the sub job. The bulk number is set in the environment variable PJM_BULKNUM in the sub job.

The following example shows a bulk job.

Figure 2.1 Example of a bulk job



The following example shows a description in the job script of the above job.

```
#!/bin/sh
IN_DATA=./indata.${PJM_BULKNUM}              (1)
OUT_DATA=./outdata.${PJM_BULKNUM}            (2)

prog -i ${IN_DATA} -o ${OUT_DATA}            (3)

* The options at the job submission time (e.g., specifying the amount of resources)
  are omitted here to simplify the description.
```

(1) Determines the input data file name with the bulk number.
(2) Determines the output data file name with the bulk number.
(3) Specifies and executes the input/output data files with arguments of the program prog.

2.1.1.2 Creating a step job

A step job determines whether to execute a new sub job according to the results of past completed sub jobs. Therefore, consider the following points when creating a job.

Then, specify the condition for execution of the next sub job according to the end code of a sub job.

For details on how to specify it, see "2.3.4.2 How to submit a step job."

2.1.1.3 Creating a workflow job

In a workflow job, the user controls job submission with a shell script.

An example is a shell script that automatically submits a job after a specific job ends or selects the next job to submit based on the results of another job.

The user can thereby control job submission with the pjwait command provided by the Job Operation Software.

- Wait for job end

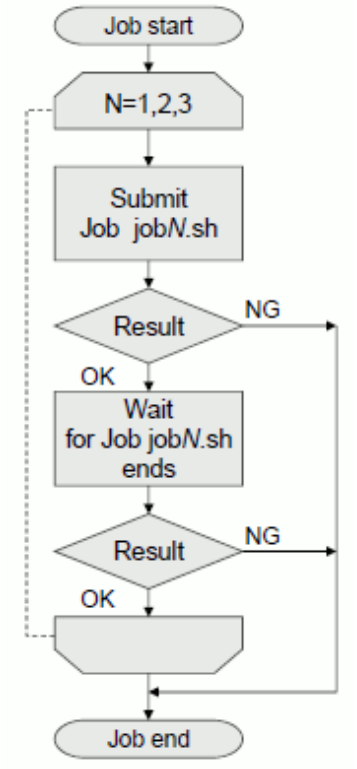
  You can use the pjwait command to wait for one or more jobs to end in a shell script.

- Result of job execution

  You can use the pjwait command to obtain the job exit code (PJM code; result code of the job manager), the exit code of a job script and the signal number when the job script ends.

The following example shows a workflow job.

Figure 2.2 Example of a workflow job



The following shows a description in the shell script of the above workflow job.

```
#!/bin/sh

for no in 1 2 3                          (1)
do
    JID=`pjsub -z jid job${no}.sh`       (2)
    if [ $? -ne 0 ]; then                (3)
        exit 1
```

```
    fi
    set -- `pjwait $JID`                    (4)
    if [ $2 != "0" -o $3 != "0"]; then      (5)
        exit 1
    fi
done                                        (6)
```

(1) Submits job scripts job1.sh, job2.sh, and job3.sh in this order.
(2) Displays the job ID with the -z option, and assign it to the shell variable JID.
(3) Ends the workflow job when the job submission result is other than 0.
(4) Waits for job end, and assigns the output to the positional parameter.

It is necessary to devise it so that the output result might become a multi-line according to the kind of the job and a method of specifying the job ID. For detail of the output, see the man page of pjwait command.

(5) Ends the workflow job when the PJM code ($2) or the job end code ($3) is other than 0.
(6) Goes to execution of the next job.

2.1.1.4 Creating a master-worker job

For details on how to create a master worker job, see "Job Operation Software End-user's Guide for Master-Worker Job."

The job operation software supports the following three methods as methods for implementing master-worker jobs, from the viewpoint of how to generate worker processes.

- Generating a worker process with MPI dynamic process generation

  This method is used when both the master process and worker process are MPI programs. That is, MPI mechanisms will be used for worker process generation and communication.
  The job operation software selects the node that will generate a worker process.

- Generating a worker process from an Agent process

  This method is used when the worker process is not an MPI program.

  Communication between processes uses a communication function such as a socket. The user controls and manages the generation of the worker process and the selection of the node generating the worker process.

- Generating a worker process with the pjaexe command

  Similar to the preceding paragraph, this method is also used when the worker process is not an MPI program.
  Likewise in this method, communication between processes uses a communication function such as a socket, and the user controls and manages the selection of the node that will generate the worker process. However, the pjaexe command provided by the job operation software is used to generate the worker process.

To create a master-worker job, the user needs to select a method matching the purpose of the job.

2.1.1.5 Creating a job for executing an MPI program

To execute an MPI program as a job, the node resources required for the MPI program must be allocated properly. Therefore, the user needs to exercise caution when writing these specifications in the job script.

For details, see "2.3.6 Submitting an MPI job."

Generally, MPI programs other than Development Studio execute the process remotely on the nodes by using rsh and the ssh command. However, use the pjrsh command instead of them to execute the MPI program on Job Operation Software.
For details on how to use the pjrsh command instead of the rsh or ssh command, see "Appendix C Executing programs of MPI processing system other than Development Studio."

Furthermore, some cases may require the setting of an environment variable specific to the MPI program. For details, see the "MPI User's Guide", which is a Development Studio manual. Development Studio is related software for creating MPI programs.

2.1.1.6 Execution a sequential program on the virtual nodes all at once [PG]

To execute a sequential program on multiple virtual nodes all at once, the pjexe command can be used.

The following example shows the execution of the sequential program prog on eight virtual nodes.

```
pjexe --vnode 8 prog
```

2.1.1.7 Creating a job that uses PAPI library or strace command [FX]

To execute the following programs in a job to be submitted to the FX server, the xospastop command (/usr/bin/xospastop) must be executed before executing the program.

- Program linked to the PAPI (Performance Application Program Interface) library of the open source software (including a program using, for example, TAU or Scalasca of the open source software)

[CODING EXAMPLE]

```
#!/bin/bash
xospastop            <- Added description
./a.out
```

If the xospastop command is not executed, the programs described above may not run normally.

When any of the following functions of Development Studio is used, it meets the above condition; however, executing the xospastop command is not required.

- Profiler
- Function for outputting information at execution

Information

The xospastop command is a command provided by the operating system for FX server.
This command stops collecting the PMU counter (Performance Monitoring Unit Counter) of a job. There is no option or message output. The end status is always 0.

2.1.1.8 Procedure for avoiding disturbance to job execution performance

Disturbances affecting job performance should be dealt with as follows.

- Disturbance by standard output/standard error output of the mpiexec command [FX]

  Collecting the standard output and standard error output of the mpiexec command into a single file can affect job execution performance. To avoid this affect, the standard output and standard error output of the mpiexec command should be printed to a separate file for each rank. For more information, see "2.3.6.9 Standard output/standard error output of the mpiexec command [FX]."

- Disturbance by collection of the job statistical information [FX]

  The periodic collection of job statistical information by the job resource management function can affect the execution performance of a job. You can avoid this disturbance by executing the xospastop command within the job. See "2.1.1.7 Creating a job that uses PAPI library or strace command [FX]" for more information.

2.1.2 Environment variables in jobs

The Job Operation Software sets the following environment variables in jobs.

Table 2.1 Environment variables that may be referenced in jobs

Environment variable	Detail
PJM_ADAPTIVE_ELAPSED_TIME_MAX [FX]	Maximum value (seconds) for the elapsed time limit of a job, as specified in the -L elapse option of the pjsub command If specified with a range, like with "-L elapse=30:00-1:00:00", the maximum value (1:00:00) is set as a number of seconds (3600). If the -L elapse option is not specified, the default value defined by the job ACL function is set. If specified like "-L elapse=30:00", this environment variable itself is not set.

Environment variable	Detail
PJM_ADAPTIVE_ELAPSED_TIME_MIN [FX]	Minimum value (seconds) for the executable time of a job, as specified in the -L elapse option of the pjsub command If the executable time is specified with a minimum value and a maximum value, like with "-L elapse=30:00-1:00:00", the minimum value (30:00) is set as a number of seconds (1800). If the -L elapse option is not specified, the default value defined by the job ACL function is set. If specified like "-L elapse=30:00", this environment variable itself is not set.
PJM_APPNAME	Character string specified in the --appname option of the pjsub command If the option is not specified, this environment variable is not set.
PJM_ASSIGN_LOGICAL_CPU [PG]	Range of logical CPUs on which job processes operate This is valid for job processes other than those of "C/C++/Fortran programs created in Development Studio." job: Job processes can use only the logical CPUs for the job in the allocated CPU core. all: Job processes can use all the logical CPUs in the allocated CPU core. For details on the logical CPUs for a job, see "C.5 Setting the Range of CPU Resources Available to a Job [PG]."
PJM_ASSIGN_ONLINE_NODE [FX]	If the --mpi assign-online-node option of the pjsub command is specified, 1 is set. If the option is not specified, this environment variable is not set.
PJM_AT	Job execution start time (*YYYY*/*MM*/*DD hh*:*mm*:*ss*) specified in the --at option of the pjsub command (*YYYY*:Year, *MM*:Month, *DD*:Day, *hh*:Hour, *mm*:Minute, *ss*:Second) If the option is not specified, this environment variable is not set.
PJM_BULKNUM	Bulk number (set only for a bulk job)
PJM_COMMENT	Character string specified in the --comment option of the pjsub command If the option is not specified, this environment variable is not set.
PJM_CORE_MEM_LIMIT [PG]	Amount of memory (bytes) per CPU core specified in the -L core-mem option of the pjsub command If the option is not specified, this environment variable is not set.
PJM_CUSTOM_RESOURCES	Custom resource name and requested quantity (or requested type) specified in the -L option of the pjsub command Example: *customrscname*=1 (*customrscname* indicates a defined custom resource name) If the option is not specified, the default value defined by the job ACL function is set. If no default value is defined by the job ACL function, 0 is set as the requested quantity. In this case, the requested type is not set. If multiple custom resources are requested, the resource information is delimited by a comma (,). If a custom resource per node is requested, "/n" is added to the custom resource name.
PJM_DPREFIX	Prefix indicating a command line from a job script to the Job Operation Software. By default, it is as set by "#PJM", but you can change it with the -C or --dir-prefix option of the pjsub command.

Environment variable	Detail
PJM_ELAPSED_TIME_MODE [FX]	Specification method for the executable time specified for a job in the -L elapse option of the pjsub command Either of the following is set: - "fixed" This is the setting when the executable time is specified like "-L elapse=30:00". - "adaptive" This is the setting when the executable time is specified with a minimum value and a maximum value, like with "-L elapse=30:00-" or "-L elapse=30:00-1:00:00". If the -L elapse option is not specified, the default value defined by the job ACL function is set.
PJM_ELAPSE_LIMIT	Maximum executable time (seconds) for the job specified in the-L elapse option of the pjsub command If the option is not specified, the default value defined by the job ACL function is set.
PJM_ENVIRONMENT	"BATCH" for a batch job, and "INTERACT" for an interactive job are set.
PJM_EXEC_POLICY [PG]	Execution mode policy specified in the -P exec-policy option of the pjsub command simplex : SIMPLEX (Occupy nodes) share : SHARE (Share nodes) If the option is not specified, the default value defined by the job ACL function is set. For a node allocation job, this environment variable is not set.
PJM_FSNAME	Character string specified in the --fs option of the pjsub command If the option is not specified, this environment variable is not set.
PJM_JOBDIR	Path of the current directory of the beginning to execute job script in the case of not using rank number directory.
PJM_JOBID	Job ID
PJM_JOBNAME	Job name
PJM_LLIO_ASYNC_CLOSE [FX]	Operation of whether or not to asynchronously close files on first-layer storage and second-layer storage on: Asynchronous close off: Synchronous close The setting of asynchronous close ("on") does not guarantee the completion of writing when a file is closed. The setting of synchronous close ("off") guarantees writing completion. This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_AUTO_READAHEAD [FX]	Operation of whether or not to automatically read ahead when a job is going to try reading consecutive areas again on first-layer storage or second-layer storage multiple times on: Read ahead. off: Do not read ahead. This environment variable is not set in systems that do not use the layered storage system.

Environment variable	Detail
PJM_LLIO_CN_CACHED_WRITE_SIZE [FX]	Threshold value of whether or not to cache data when writing the data to first-layer storage (Bytes) When writing data to first-layer storage, if the write size is equal to or less than this value, the data is not immediately written to the storage. Instead, it is temporarily cached in compute node cache. This parameter improves performance by outputting smaller data volumes collectively, reducing the number of data transfers to first-layer storage. This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_CN_CACHE_SIZE [FX]	Compute node cache size on first-layer storage (Bytes) This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_CN_READ_CACHE [FX]	Operation of whether or not to cache a file in compute node cache when reading the file from first-layer storage or second-layer storage on: Cache. off: Do not cache. This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_LOCALTMP_SIZE [FX]	Size of the node temporary area on first-layer storage (Bytes) This environment variable is not set in systems that do not use the layered storage system. The path to the node temporary area is set in the environment variable PJM_LOCALTMP.
PJM_LLIO_PERF [FX]	Operation of whether or not to output the LLIO performance information file on: Output. off: Do not output. This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_PERF_PATH [FX]	Path to the LLIO performance information file This setting applies only to output of the LLIO performance information file. This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_SHAREDTMP_SIZE [FX]	Size of the shared temporary area on first-layer storage (Bytes) (*) Multiply this size by the number of allocated compute nodes, and the resulting value is the size of the shared temporary area available to one job. This environment variable is not set in systems that do not use the layered storage system. The path to the shared temporary area is set in the environment variable PJM_SHAREDTMP.
PJM_LLIO_SIO_READ_CACHE [FX]	Operation of whether or not to cache a file in first-layer storage when reading the file from second-layer storage to a compute node on: Cached. off: Do not cache.

Environment variable	Detail
	This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_STRIPE_COUNT [FX]	Number of stripes per file when files are distributed across first-layer storage This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_STRIPE_SIZE [FX]	Stripe size when files are distributed across first-layer storage (Bytes) This environment variable is not set in systems that do not use the layered storage system.
PJM_LLIO_UNCOMPLETED_FILEINFO_PATH [FX]	Output path for a list of names of files that have not completed writing The file name list is output to the file in this path when writing to second-layer storage has not completed at job end. The listed files are still in the cache. This environment variable is not set in systems that do not use the layered storage system.
PJM_LOCALTMP [FX]	Path to the node temporary area on first-layer storage This environment variable is not set in systems that do not use the layered storage system.
PJM_MAILOPTION	Operation specified with the -m option of the pjsub command for mail notification Logical OR of the following value (hexadecimal) is set. 0x1 : Send e-mail at beginning of job execution (-m b) 0x2 : Send e-mail at end of job execution (-m e) 0x4 : Send e-mail at restart of the job (-m r) 0x8 : Inform job statistical information (-m s) 0x10 : Inform job statistical information and node statistical information (-m S) Example: PJM_MAILOPTION=0x3 If the option is not specified, this environment variable is not set.
PJM_MAILSENDTO	User specified as the e-mail destination in the --mail-list option of the pjsub command If the option is not specified, this environment variable is not set.
PJM_MPI_PROC	Maximum number of processes that are generated at the start of the MPI program specified in the --mpi proc option of the pjsub command
PJM_MPI_SHAPE_X PJM_MPI_SHAPE_Y PJM_MPI_SHAPE_Z [FX]	Number of nodes in the X-, Y-, and Z-axis directions for the process shape specified in the --mpi shape option of the pjsub command If the option is not specified, the number of nodes for the node shape specified in the -L node option of the pjsub command is used. In addition, environment variables that do not correspond to the dimensions of the shape are not set. For example, the environment variables PJM_MPI_SHAPE_Y and PJM_MPI_SHAPE_Z are not set for a one-dimensional shape.
PJM_NET_ROUTE [FX]	Operation when a link goes down for the Tofu interconnect used as the communication path by a job dynamic : Change the communication path, and job execution continues. static : Do not change the communication path. The job ends abnormally.
PJM_NODE	Value specified in the -L node option of the pjsub command (For FX servers, the value is the number of nodes determined by the specified node

Environment variable	Detail
	shape.) If the option is not specified, the default value set by the job ACL function is used (which is "(node=)" at "pjsub option parameters" displayed by the pjacl command as shown in "2.2.2 Checking restriction information"). This environment variable is set only when physical nodes are allocated, and it is not set when virtual nodes are allocated.
PJM_NODE_ALLOCATION_IO_MODE [FX]	Node allocation method related to the exclusive I/O specified in the -L node option of the pjsub command. io-exclusive : I/O-exclusive mode no-io-exclusive : I/O-sharing mode If the option is not specified, the default value defined by the job ACL function is set.
PJM_NODE_ALLOCATION_MODE [FX]	Node allocation method specified in the -L node option of the pjsub command torus : torus mode mesh : mesh mode noncont : non-contiguous mode If the option is not specified, the default value defined by the job ACL function is set.
PJM_NODE_MEM_LIMIT [FX]	Upper limit of memory usage (bytes) per node specified in the -L node-mem option of the pjsub command If the option is not specified, the default value defined by the job ACL function is set.
PJM_NODE_X PJM_NODE_Y PJM_NODE_Z [FX]	Number of nodes in the X-, Y-, and Z-axis directions for the shape specified in the -L node option of the pjsub command In addition, environment variables that do not correspond to the dimensions of the shape are not set. For example, the environment variables PJM_NODE_Y and PJM_NODE_Z are not set for a one-dimensional shape.
PJM_NORESTART	If the --norestart option of the pjsub command is specified, 1 is set. If the option is not specified, this environment variable is not set.
PJM_O_HOME	Environment variable HOME of the user who executed the pjsub command
PJM_O_HOST	Name of the host that executed the pjsub command
PJM_O_LANG	Environment variable LANG of the user who executed the pjsub command
PJM_O_LOGNAME	Environment variable LOGNAME of the user who executed the pjsub command
PJM_O_MAIL	Environment variable MAIL of the user who executed the pjsub command
PJM_O_NODEINF [PG]	Path of the allocated node list file For a job to which virtual nodes are allocated, the IP addresses of the physical nodes where the virtual nodes are placed are written one per line.
PJM_O_PATH	Environment variable PATH of the user who executed the pjsub command
PJM_O_SHELL	Environment variable SHELL of the user who executed the pjsub command
PJM_O_TZ	Environment variable TZ of the user who executed the pjsub command
PJM_O_WORKDIR	Current directory at the pjsub command execution time
PJM_PROC_BY_NODE	Maximum number of processes that are generated per physical node or virtual node by an MPI program

Environment variable	Detail
	However, if a physical node (node=1) or virtual node (vnode=1) is allocated and the -mpi option of the pjsub command is not specified, this environment variable is not set.
PJM_RANK_MAP_BYCHIP	Rank allocation rule specified in the --mpi rank-map-bychip option of the pjsub command - FX servers The XY, YX, XYZ, XZY, YXZ, YZX, ZXY, or ZYX character string is set. - PRIMERGY servers The number of virtual nodes to be allocated per physical node is set. If the option is not specified, this environment variable is not set. If no argument of the option is specified, DEF is displayed for the environment variable.
PJM_RANK_MAP_BYNODE	Rank allocation rule specified in the --mpi rank-map-bynode option of the pjsub command - FX servers The XY, YX, XYZ, XZY, YXZ, YZX, ZXY, or ZYX character string is set. - PRIMERGY servers If the option is specified, 1 is set. If the option is not specified, this environment variable is not set. If no argument of the option is specified, DEF is displayed for the environment variable.
PJM_RSCGRP	Resource group name specified in the -L rscgrp or -L rg option of the pjsub command If the option is not specified, the default value defined by the job ACL function is set.
PJM_RSCUNIT	Resource unit name specified in the -L rscunit or -L ru option of the pjsub command If the option is not specified, the default value defined by the job ACL function is set.
PJM_SHAREDTMP [FX]	Path to the shared temporary area on first-layer storage This environment variable is not set in systems that do not use the layered storage system.
PJM_SHELL	Path of the job execution shell
PJM_STDERR_PATH	Path of the standard error output file for jobs as specified in the -e option of the pjsub command If the option is not specified, the default file name "*job name*.e.*job ID*" is set. For bulk and step jobs, however, "*job ID*" contains a sub job ID. If the -j option of the pjsub command is specified, the same value as the environment variable PJM_STDOUT_PATH is used.
PJM_STDOUT_PATH	Path of the standard output file for jobs as specified in the -o option of the pjsub command If the option is not specified, the default file name "*job name*.o.*job ID*" is set. For bulk and step jobs, however, "*job ID*" contains a sub job ID.
PJM_STEPNUM	Step number (set only for a step job)
PJM_SUBJOBID	Sub job ID

Environment variable	Detail
	A job ID is set for a normal job.
PJM_VNODE	Value specified in the -L vnode option of the pjsub command If the option is not specified, the set default value in an item of the job ACL function is used (which is "(vnode=)" at "pjsub option parameters" displayed by the pjacl command as shown in "2.2.2 Checking restriction information"). This environment variable is set only when virtual nodes are allocated, and it is not set when physical nodes are allocated.
PJM_VNODE_CORE	Value specified in the -L vnode-core or -L vnode=(core=) option of the pjsub command If the option is not specified, the set default value in an item of the job ACL function is used (which is "(vnode-core=)" at "pjsub option parameters" displayed by the pjacl command as shown in "2.2.2 Checking restriction information"). This environment variable is set only when virtual nodes are allocated, and it is not set when physical nodes are allocated.
PJM_VNODE_MEM_LIMIT [PG]	Amount of memory (bytes) per virtual node specified in the -L vnode-mem or -L mem option of the pjsub command If the option is not specified, the default value defined by the job ACL function is set.
PJM_VN_POLICY [PG]	Virtual node placement policy specified in the -P vn-policy option of the pjsub command abs-pack : Absolutely PACK pack : PACK abs-unpack : Absolutely UNPACK unpack : UNPACK If the option is not specified, the default value defined by the job ACL function is set.
Others	Environment variables which are specified with -x or -X option of the pjsub command

Note

- Even when the -x option of the pjsub command specifies an environment variable with the same name as an environment variable shown in "Table 2.1 Environment variables that may be referenced in jobs," its value is ignored. The variable takes the value shown in "Table 2.1 Environment variables that may be referenced in jobs."
- Unless an instruction is given, a set environment variable on the shell executing the pjsub command will not be passed within a job. To pass the environment variable within the job, specify the -X option of the pjsub command.
  However, even when the -X option of the pjsub command specifies an environment variable with the same name as an environment variable shown in "Table 2.1 Environment variables that may be referenced in jobs," its value is ignored. The variable takes the value shown in "Table 2.1 Environment variables that may be referenced in jobs."
- End users should not use environment variables that begin with "PJM_PK_". Environment variables beginning with "PJM_PK_" are used by the power knob operation. For the power knob operation function, refer to "Job Operation Software API user's Guide for Power API."
- Environment variables beginning with "PJM_LLIO_" are used by LLIO operation. For LLIO operation function, refer to "LLIO User's Guide." End users should not use environment variables that begin with "PJM_LLIO_" that ar not listed in "Table 2.1 Environment variables that may be referenced in jobs."
- Environment variable LD_LIBRARY_PATH in a job
  You need to be careful when setting the current directory on the FEFS file system in the environment variable LD_LIBRARY_PATH within a job.
  Unless the shared library required for program execution is placed in the current directory, do not include the current directory in the

environment variable LD_LIBRARY_PATH.
If the search path includes the current directory on the FEFS file system, unnecessary access to files increases, which may result in a high load on the FEFS file system.

The following example shows settings specifying that the current directory be searched.

```
a) LD_LIBRARY_PATH=/usr/local/lib:
b) LD_LIBRARY_PATH=/usr/local/lib:.
c) LD_LIBRARY_PATH=/usr/local/lib:.:/usr/lib
d) LD_LIBRARY_PATH=:/usr/local/lib
e) LD_LIBRARY_PATH=/usr/local/lib::/usr/lib
```

Examples a, d, and e above do not explicitly specify the current directory but instead make it a search target.
In these cases, set one of the following:

```
a or d) LD_LIBRARY_PATH=/usr/local/lib
e) LD_LIBRARY_PATH=/usr/local/lib:/usr/lib
```

Then, the current directory is not a search target.

2.1.3 How to create a user program for execution

For details on how to create programs, see the manual provided with Development Studio. For details on how to create MPI programs to be executed within jobs, automatic parallelization programs with the compiler, and other such programs, see the same manual.

2.1.4 File systems available to jobs

The job scripts, user-created programs, and data files necessary for job execution are placed on the login node where the jobs are submitted. However, the compute nodes executing the jobs must also be able to access these files. Therefore, place them on a shared file system that can be accessed on the same path by the login node and compute nodes.

When creating a file as job execution results, output it on this shared file system. Then, after the job outputs the file on a compute node, you can view the file on the login node. The path to the shared file system will vary depending on system. For details, contact the administrator.

[Layered storage system]

You can use the node temporary area and shared temporary area of first-layer storage as high-speed storage for jobs in systems where the layered storage system is available. These areas are secured for each job, so they are available only to that job. You can find the paths to the node temporary area and shared temporary area in the environment variables PJM_LOCALTMP and PJM_SHAREDTMP, which are set in each job.

The following example shows a simple job using the node temporary area as the temporary data storage destination.

1. The program prog1 outputs the calculation results out.data to the node temporary area ${PJM_LOCALTMP}.

   ```
   prog1 -o ${PJM_LOCALTMP}/out.data
   ```

2. The program prog2 reads the out.data file and then outputs the final calculation results result.data to the ${PJM_LOCALTMP} directory.

   ```
   prog2 -i ${PJM_LOCALTMP}/out.data -o ${PJM_LOCALTMP}/result.data
   ```

3. Before the job ends, change the name of the result.data file to a name with the job ID ${PJM_JOBID}, and save the file to the job execution directory ${PJM_JOBDIR} (on the global file system).

   ```
   cp ${PJM_LOCALTMP}/result.data ${PJM_JOBDIR}/ result_${PJM_JOBID}.data
   ```

Files are output to the node temporary area and shared temporary area. If necessary, save the files to second-layer storage before the job ends. The node temporary area and shared temporary area are temporary areas deleted after the job ends.

If the job contains a program to access files on second-layer storage, copy these files to the second-layer storage cache with the llio_transfer command before the program is executed in the job. This can be expected to improve file access performance. The command is provided by LLIO. For details on the llio_transfer command, see the "Reference" appendix in "LLIO User's Guide."

The following example shows the llio_transfer command used in a job.

```
#!/bin/bash
#PJM -L "node=2"
#PJM --llio sharedtmp-size=5Mi,localtmp-size=10Gi
llio_transfer /gfs1/user1/a.out          <- Copy a.out file on second-layer storage cache
mpiexec /gfs1/user1/a.out                <- Execute a.out program
llio_transfer --purge /gfs1/user1/a.out  <- Delete a.out file from second-layer storage cache
```

Note

Files copied to the second-layer storage cache using the llio_transfer command are called common files. Note the following about common files:

- Deleting common files, or updating their data or attributes fail with an error via the cache area of the second-layer storage.
- File locking operations for common files are not supported.
- During job execution, do not change or delete the copy source file on the second-layer storage. If the file is changed, the accuracy of the file contents copied to the second-layer storage cache cannot be guaranteed. Also, if it is deleted, the common file cannot be deleted with the --purge option of the llio_transfer command, and the cache space of the second-layer storage remains occupied by the common file until the job is finished.
- To delete a common file in a job script, specify the --purge option in the llio_transfer command. The rm command against the common file fails and it cannot be deleted.
- Shortly after deleting common files with llio_transfer --purge, operations to delete the copy source files or update their data or attributes may fail with an error. If this occurs, wait for 60 seconds, and then retry the operations.
- The "cache area of second-layer storage" located at the first-layer storage, which is the common file copy destination, must have enough space to store the entire common file. Common files copied to a cache area of second-layer storage are not deleted when this area is full. If you need space in the cache area of the second-layer storage, use the llio_transfer command with the --purge option to remove common files from the cache area during the job. Note that the common file in the cache area of the second-layer storage is automatically deleted after the job ends.
- If the cache data for the transfer source file exists in the first-layer storage before the command allocates the common file area in the job, the llio_transfer command fails. Do not open the file in the job before executing the llio_transfer command because the cache data may be created when the file is opened. Do not use the following commands, as they may also open files.
  - lfs setstripe
  - lfs getstripe

  Note that if a job continues with failure of the llio_transfer command, the job executes with no common files deployed.
- Files that can be specified in the llio_transfer command must meet the following conditions.
  - A file that can be accessed by the user executing this command, and
  - A file in the second-layer storage, and
  - A file type of a regular file, and
  - A file size of 1 byte or more

2.2 Checking Job-related Information

This section describes the information that users need to confirm before submitting their jobs.

2.2.1 Checking resource units and resource groups

To submit a job, the user checks whether the resource groups and resource units for executing the job satisfy size and shape requirements.

The user also checks the status of control of job submission to the resource units and resource groups, since the administrator may have such control.

You can check information about resource units and resource groups by using the --rsc option of the pjstat command.

```
$ pjstat --rsc
RSCUNIT                        RSCUNIT_SIZE  RSCGRP                      RSCGRP_SIZE
unit1[ENABLE,START]            20x9x16       group1[ENABLE,START]        20x3x16
unit1[ENABLE,START]            20x9x16       group2[ENABLE,STOP]         20x6x16
unit2[ENABLE,STOP]             20x9x16       group1[ENABLE,START]        1500
unit3[DISABLE,STOP]            4x18x16       group1[ENABLE,START]        1152
```

Table 2.2 pjstat command display items (1)

Item	Description
RSCUNIT	Name and state of a resource unit Format: *resource unit name*[*state*,...] "*state*" means the following. ENABLE: Job can be submitted DISABLE: Job cannot be submitted START: Job can be executed STOP: Job cannot be executed
RSCUNIT_SIZE	Size of a resource unit - FX servers The size is expressed as a cuboid with each dimension representing the number of Tofu units along the X, Y, or Z axis. Format: *X*x*Y*x*Z* - PRIMERGY servers The number of nodes *N* that compose the resource unit is displayed.
RSCGRP	Name and state of a resource group Format: *resource group name*[*state*,...] The meaning of "*state*" is the same as for the previous item, RSCUNIT.
RSCGRP_SIZE	Size of a resource group - FX servers The size is expressed by a cuboid, with each dimension representing a number of nodes, or by a number of nodes. Format: *X*x*Y*x*Z* (cuboid) or *N* (number of nodes) - PRIMERGY servers The number of nodes *N* that compose the resource group is displayed.

Ask the administrator about the compute node models for resource units and resource groups.

The number of nodes allocated to the job and the node shape must fit within the resource group size (RSCGRP_SIZE shown above). If it exceeds this size, there will not be enough nodes to execute the job.

However, not all of the nodes shown by the RSCGRP_SIZE item in a resource group of FX servers may be allocable to the job, depending on the configuration.
The largest shape of nodes that can be allocated to the job can be obtained from the MAX_SIZE item, which is displayed by the pjstat command with the --shape option. See "Table 2.4 Size of nodes that can be allocated [FX]."

```
$ pjstat --rsc --shape
RSCUNIT                     RSCUNIT_SIZE    RSCGRP                    RSCGRP_SIZE   MAX_SIZE
rscunit000[ENABLE,START]    3x2x9           rscgroup1[ENABLE,START]   540           4x6x18
```

```
rscunit000[ENABLE,START]     3x2x9          rscgroup2[ENABLE,START]  2x3x18        2x3x18
rscunit000[ENABLE,START]     3x2x9          rscgroup3[ENABLE,START]  2x3x18        2x3x18
rscunit000[ENABLE,START]     3x2x9          rscgroup4[ENABLE,START]  540           4x6x18

* The above is an example for a resource group of FX servers.
```

Table 2.3 pjstat command display items (2)

Item	Description
MAX_SIZE	Maximum size of nodes that can be allocated to a job - FX servers The size is expressed by a cuboid, *X*x *Y*x*Z*, with each side representing the number of nodes. If there are multiple largest shapes, the command displays all of them, delimited by a comma. - PRIMERGY servers The displayed number of nodes, *N*, is equal to the RSCGRP_SIZE item.

Table 2.4 Size of nodes that can be allocated [FX]

Node shape	Size of allocable nodes
One-dimensional shape	- In torus mode or mesh mode The number of nodes must be up to the product (*X** *Y***Z*) of the shape indicated by the item MAX_SIZE. If the RSCGRP_SIZE item is displayed as the number of nodes, *N*, the number of nodes must be *N* or less. - In non-contiguous mode The number of nodes must be up to the product (*X** *Y***Z*) of the shape indicated by the item RSCGRP_SIZE or up to the specified number (*N*) of nodes. The item MAX_SIZE is not applicable in the case of non-contiguous mode.
Two-dimensional shape	- In torus mode or mesh mode The nodes must fit within the shape (*X**3)x(*Y***Z*/3) obtained from the item MAX_SIZE (*X*x *Y*x*Z*), or any shape obtained as the result of rotating the shape. If the RSCGRP_SIZE item is displayed as the number of nodes, *N*, the number of nodes must be *N* or less. - In non-contiguous mode The number of nodes must be up to the product (*X** *Y***Z*) of the shape indicated by the item RSCGRP_SIZE or up to the specified number (*N*) of nodes. The item MAX_SIZE is not applicable in the case of non-contiguous mode.
Three-dimensional shape	- In torus mode or mesh mode The nodes must fit within the shape *X*x *Y*x*Z* indicated by the item MAX_SIZE, or any shape obtained as the result of rotating the shape. If the RSCGRP_SIZE item is displayed as the number of nodes, *N*, the number of nodes must be *N* or less. - In non-contiguous mode The number of nodes must be up to the product (*X** *Y***Z*) of the shape indicated by the item RSCGRP_SIZE or up to the specified number (*N*) of nodes. The item MAX_SIZE is not applicable in the case of non-contiguous mode.

[Example]

In the above display example, the number of nodes in the resource group rscgroup1 is 540. However, the maximum size of nodes that can be allocated is obtained from the MAX_SIZE item (4 x 6 x 18) as follows:

- For one-dimensional shape
  In torus mode or mesh mode, the number of nodes must be up to RSCGRP_SIZE = 540 and up to MAX_SIZE = 4 x 6 x 18 = 432.
  That is to say, the maximum number of nodes is 432.
  In non-contiguous mode, the number of nodes must be up to RSCGRP_SIZE = 540.

- For two-dimensional shape
  In torus mode or mesh mode, nodes must fit within the maximum shape of (4*3) x (6*18/3) = 12 x 36 or any shape obtained as the result of rotating the shape: 36x12. In addition, the number of nodes must be up to RSCGRP_SIZE = 540. That is to say, the maximum number of nodes, which is restricted by the maximum shape, is 12 x 36 = 432.
  In non-contiguous mode, the number of nodes must be up to RSCGRP_SIZE = 540.

- For three-dimensional shape
  In torus mode or mesh mode, nodes must fit within the maximum shape of 4 x 6 x 18 or any of the following shapes obtained as a result of rotating the maximum shape: 4 x 18 x 6, 6 x 4 x 18, 6 x 18 x 4, 18 x 6 x 4, or 18 x 4 x 6. (If ":strict" is specified, the nodes must fit within 4 x 6 x 18 only.) In addition, the number of nodes must be up to RSCGRP_SIZE = 540. That is to say, the maximum number of nodes, which is restricted by the maximum shape, is 4 x 6 x 18 = 432.
  In non-contiguous mode, the number of nodes must be up to RSCGRP_SIZE = 540.

2.2.2 Checking restriction information

The job ACL function restricts use of the system by users. (See "1.10 Job ACL Function.") Users need to submit and execute jobs within the restrictions.

You can use the pjacl command to check the information on restrictions that apply to a user.

```
$ pjacl
#
# JOBACL information
#
user    user1                                          (1)
group   group1                                         (2)

defines                                                (3)
    default rscunit             rscunit1               (4)
    default rscgroup            rscgroup1              (5)
    default rscunit(interact)   rscunit1               (6)
    default rscgroup(interact)  rscgroup1              (7)
    default alloc granularity   node                   (8)
    default elapsed time mode   fixed                  (9)
    user priority               127                    (10)
    fairshare init              0                      (11)
    fairshare recovery          100                    (12)
    ingroup priority            128                    (13)
    ingroup fairshare init      100                    (14)
    ingroup fairshare recovery  1                      (15)
    node-priority               31                     (16)
    numa-policy                 pack                   (17)
    default allocation-mode     torus                  (18)
    default allocation-io-mode  no-io-exclusive        (19)
    default strict-mode         nostrict               (20)
    net-route                   dynamic                (21)
    load-policy                 balancing              (22)
    vn-policy                   pack                   (23)
    exec-policy                 share                  (24)
    assign-logical-cpu          job                    (25)
    pjstat display mode         anonymous              (26)
    mpiexec-stdouterr-unit      mpiexec                (27)
    mpiexec-stdout              %o                     (28)
    mpiexec-stderr              %e                     (29)
    mpiexec-stdout(interact)    noset                  (30)
    mpiexec-stderr(interact)    noset                  (31)
    mpiexec-std-emptyfile       on                     (32)
    default llio-async-close    on                     (33)
    default llio-auto-readahead on                     (34)
    default llio-cn-read-cache  on                     (35)
    default llio-sio-read-cache off                    (36)
    default llio-uncompleted-fileinfo-path  %e         (37)
    default llio-perf           off                    (38)
```

```
    default llio-perf-path      %n.%J.llio_perf             (39)

groups                                                      (40)
    group priority              127                         (41)
    group fairshare init        0                           (42)
    group fairshare recovery    100                         (43)

pjsub option parameters                                                 (44)
    (-L/--rsc-list)                     lower      upper           default       (45)
        (elapse=)                       1          24:00:00        01:00:00      (46)
        (adaptive elapsed time min)     00:00:01   24:00:00        01:00:00      (47)
        (adaptive elapsed time max)     00:00:02   48:00:00        02:00:00      (48)
        (node elapse)                   1          unlimited       -             (49) (*1)
        (adaptive node elapse min)      00:00:01   unlimited       -             (50) (*1)
        (adaptive node elapse max)      00:00:02   unlimited       -             (51) (*1)
        (total cores elapse)            1          unlimited       -             (52) (*1)
        (total cores)                   1          unlimited       -             (53) (*1)
        (node=)                         1          2147483647      2             (54)
        (node-mem=)                     1          unlimited       unlimited     (55)
        (vnode=)                        1          2147483647      1             (56)
        (vnode-core=)                   1          2147483647      1             (57)
        (vnode-mem=)                    1          unlimited       unlimited     (58)
        (proc-core=)                    -          unlimited       0             (59)
        (proc-cpu=)                     -          unlimited       unlimited     (60)
        (proc-crproc=)                  -          512             512           (61)
        (proc-data=)                    -          unlimited       unlimited     (62)
        (proc-lockm=)                   -          unlimited       unlimited     (63)
        (proc-msgq=)                    -          unlimited       unlimited     (64)
        (proc-openfd=)                  -          unlimited       1024          (65)
        (proc-psig=)                    -          unlimited       unlimited     (66)
        (proc-filesz=)                  -          unlimited       unlimited     (67)
        (proc-stack=)                   -          unlimited       unlimited     (68)
        (proc-vmem=)                    -          unlimited       unlimited     (69)
        (customrscname1=)               0          unlimited       0             (70) (*2)
        (custom-total-customrscname2=)  0          unlimited       -             (71) (*2)
        (custom-node-customrscname2=)   0          unlimited       0             (72) (*2)
        (custom-vnode-customrscname2=)  0          unlimited       0             (73) (*2)

    (-L/--rsc-list)                     select     default                       (74)
        (customrscname3=)               a,b,c      -                             (75) (*2)
        (customrscname4=)               a,b,c      -                             (*2)

    (--interact -L)                     lower      upper           default       (76)
        (elapse=)                       1          24:00:00        01:00:00
        (adaptive elapsed time min)     00:00:01   24:00:00        01:00:00
        (adaptive elapsed time max)     00:00:02   48:00:00        02:00:00
        (node elapse)                   1          unlimited       -             (*1)
        (adaptive node elapse min)      00:00:01   unlimited       -             (*1)
        (adaptive node elapse max)      00:00:02   unlimited       -             (*1)
        (total cores elapse)            1          unlimited       -             (*1)
        (total cores)                   1          unlimited       -             (*1)
        (node=)                         1          2147483647      1
        (node-mem=)                     1          unlimited       unlimited
        (vnode=)                        1          2147483647      1
        (vnode-core=)                   1          2147483647      1
        (vnode-mem=)                    1          unlimited       unlimited
        (proc-core=)                    -          unlimited       0
        (proc-cpu=)                     -          unlimited       unlimited
        (proc-crproc=)                  -          512             512
        (proc-data=)                    -          unlimited       unlimited
        (proc-lockm=)                   -          unlimited       unlimited
        (proc-msgq=)                    -          unlimited       unlimited
        (proc-openfd=)                  -          unlimited       1024
```

```
        (proc-psig=)                    -          unlimited        unlimited
        (proc-filesz=)                  -          unlimited        unlimited
        (proc-stack=)                   -          unlimited        unlimited
        (proc-vmem=)                    -          unlimited        unlimited
        (customrscname1=)               0          unlimited        0              (*2)
        (custom-total-customrscname2=) 0           unlimited        -              (*2)
        (custom-node-customrscname2=)  0           unlimited        0              (*2)
        (custom-node-customrscname2=)  0           unlimited        0              (*2)

    (-p)                                lower            upper            default  (77)
                                        -                255              127

    (--bulk --sparam)                   lower            upper            default  (78)
        (number of subjob)              -                unlimited        -

    (--llio)                            lower            upper            default  (79)
        (sharedtmp-size=)               0Mi              2147483647Mi     0Mi      (80)
        (localtmp-size=)                0Mi              2147483647Mi     0Mi      (81)
        (cn-cached-write-size=)         0Mi              2147483647Mi     4Mi      (82)
        (cn-cache-size=)                4Mi              2147483647Mi     128Mi    (83)
        (stripe-count=)                 1                2147483647       2147483647 (84)
        (stripe-size=)                  64Ki             4194240Ki        2048Ki   (85)

execute                                                                     (86)
    pjsub                       enable                                      (87)
    pjsub(--at)                 enable                                      (88)
    pjsub(batch)                enable                                      (89)
    pjsub(--interact)           enable                                      (90)
    pjsub(normal)               enable                                      (91)
    pjsub(--step)               enable                                      (92)
    pjsub(--bulk)               enable                                      (93)
    pjsub(--mswk)               disable                                     (94)
    pjsub(node)                 enable                                      (95)
    pjsub(vnode)                disable                                     (96)
    pjstat                      enable                                      (97)
    pjdel                       enable                                      (98)
    pjdel(--enforce)            disable                                     (99)
    pjdel(--no-history)         disable                                     (100)
    pjhold                      enable                                      (101)
    pjhold(--enforce)           disable                                     (102)
    pjrls                       enable                                      (103)
    pjwait                      enable                                      (104)
    pjalter                     enable                                      (105)
    pjsig                       enable                                      (106)
    pjacl                       enable                                      (107)
    mpiexec(--std)              enable                                      (108)
    mpiexec(--stdout)           enable                                      (109)
    mpiexec(--stderr)           enable                                      (110)
    mpiexec(--std)(interact)    enable                                      (111)
    mpiexec(--stdout)(interact) enable                                      (112)
    mpiexec(--stderr)(interact) enable                                      (113)
    pjsub(torus)                enable                                      (114)
    pjsub(mesh)                 enable                                      (115)
    pjsub(noncont)              enable                                      (116)
    pjsub(nostrict)             enable                                      (117)
    pjsub(strict)               enable                                      (118)
    pjsub(strict-io)            disable                                     (119)
    pjsub(no-io-exclusive)      enable                                      (120)
    pjsub(io-exclusive)         enable                                      (121)
    pjsub(-P vn-policy)         enable                                      (122)
    pjsub(-P exec-policy)       enable                                      (123)
    pjsub(fixed elapsed time)   enable                                      (124)
    pjsub(adaptive elapsed time) disable                                    (125)
```

```
    pjsub(--net-route)          disable                     (126)
    command-api                 disable                     (127)

permit                                                      (128)
    pjsub           allow own                               (129)
    pjstat          allow own                               (130)
    pjdel           allow own                               (131)
    pjhold          allow own                               (132)
    pjrls           allow own                               (133)
    pjwait          allow own                               (134)
    pjalter         allow own                               (135)
    pjsig           allow own                               (136)
    pjacl           allow own                               (137)
    pmerls          deny all                                (138)
    pmalter         deny all                                (139)
    pjshowrsc       allow own                               (140)

limit in rscunit (each users)                               (141)
    acceptable job                unlimited                 (142)
    acceptable all-subjob         unlimited                 (143)
    acceptable bulk-subjob        unlimited                 (144)
    acceptable step-subjob        unlimited                 (145)
    running job                   unlimited                 (146)
    running bulk-subjob           unlimited                 (147)
    use node                      unlimited                 (148)
    use core                      unlimited                 (149)
    acceptable job(interact)      unlimited                 (150)
    running job(interact)         unlimited                 (151)
    use node(interact)            unlimited                 (152)
    use core(interact)            unlimited                 (153)
    customrscname1                unlimited                 (154) (*2)
    customrscname1(interact)      unlimited                 (155) (*2)

limit in rscunit (total users in samegroup)                 (156)
    acceptable job                unlimited
    acceptable all-subjob         unlimited
    acceptable bulk-subjob        unlimited
    acceptable step-subjob        unlimited
    running job                   unlimited
    running bulk-subjob           unlimited
    use node                      unlimited
    use core                      unlimited
    acceptable job(interact)      unlimited
    running job(interact)         unlimited
    use node(interact)            unlimited
    use core(interact)            unlimited
    customrscname1                unlimited                 (*2)
    customrscname1(interact)      unlimited                 (*2)

limit in rscunit (total all users)                          (157)
    acceptable job                unlimited
    acceptable all-subjob         unlimited
    acceptable bulk-subjob        unlimited
    acceptable step-subjob        unlimited
    running job                   unlimited
    running bulk-subjob           unlimited
    use node                      unlimited
    use core                      unlimited
    acceptable job(interact)      unlimited
    running job(interact)         unlimited
    use node(interact)            unlimited
    use core(interact)            unlimited
    customrscname1                unlimited                 (*2)
```

```
    customrscname1(interact)      unlimited                 (*2)

limit in rscgroup (each users)                              (158)
    acceptable job                unlimited
    acceptable all-subjob         unlimited
    acceptable bulk-subjob        unlimited
    acceptable step-subjob        unlimited
    running job                   unlimited
    running bulk-subjob           unlimited
    use node                      unlimited
    use core                      unlimited
    acceptable job(interact)      unlimited
    running job(interact)         unlimited
    use node(interact)            unlimited
    use core(interact)            unlimited
    customrscname1                unlimited                 (*2)
    customrscname1(interact)      unlimited                 (*2)

limit in rscgroup (total users in samegroup)                (159)
    acceptable job                unlimited
    acceptable all-subjob         unlimited
    acceptable bulk-subjob        unlimited
    acceptable step-subjob        unlimited
    running job                   unlimited
    running bulk-subjob           unlimited
    use node                      unlimited
    use core                      unlimited
    acceptable job(interact)      unlimited
    running job(interact)         unlimited
    use node(interact)            unlimited
    use core(interact)            unlimited
    customrscname1                unlimited                 (*2)
    customrscname1(interact)      unlimited                 (*2)

limit in rscgroup (total all users)                         (160)
    acceptable job                unlimited
    acceptable all-subjob         unlimited
    acceptable bulk-subjob        unlimited
    acceptable step-subjob        unlimited
    running job                   unlimited
    running bulk-subjob           unlimited
    use node                      unlimited
    use core                      unlimited
    acceptable job(interact)      unlimited
    running job(interact)         unlimited
    use node(interact)            unlimited
    use core(interact)            unlimited
    customrscname1                unlimited                 (*2)
    customrscname1(interact)      unlimited                 (*2)
```

The meanings of the output contents are as follows.

No.	Description
1	Execution user name
2	Execution group name
3	User-related setting values
4	Name of the default resource unit for submitting a batch job
5	Name of the default resource group for submitting a batch job
6	Name of the default resource unit for submitting an interactive job

No.	Description
7	Name of the default resource group for submitting an interactive job
8	Granularity in node resource allocation
9	Format of the executable time specification for the job - fixed: Specifies the executable time. The (elapse=) item of the pjsub option parameters is used for the default value of the executable time. - adaptive: Specifies a minimum value and maximum value for the executable time. The (adaptive elapsed time min) item and (adaptive elapsed time max) item of the pjsub option parameters are used for the default minimum and maximum values, respectively, of the executable time.
10	User priority
11	Initial fair share value of the user
12	Fair share value recovery rate of the user
13	User priority in the group
14	User's initial fair share value in the group
15	User's fair share value recovery rate in the group
16	Node priority [PG]
17	NUMA allocation policy
18	Node allocation method [FX]
19	Whether the node allocation to the job is in I/O-exclusive mode.[FX]
20	Strict mode at node allocation [FX]
21	Whether to dynamically change the communication path when a link goes down for the Tofu interconnect used by a job [FX]
22	Job distributed allocation or localized allocation [PG]
23	Virtual node placement policy [PG]
24	Execution mode (node) [PG]
25	Range of logical CPUs that can be used by job processes [PG]
26	Access restriction for the job information displayed by the pjstat command
27	Output units for the standard output/standard error output of the mpiexec command [FX]
28	Default output destination for the standard output of the mpiexec command in a batch job [FX]
29	Default output destination for the standard error output of the mpiexec command in a batch job [FX]
30	Default output destination for the standard output of the mpiexec command in an interactive job [FX]
31	Default output destination for the standard error output of the mpiexec command in an interactive job [FX]
32	Whether to create an empty file if there is no standard output/standard error output for the mpiexec command [FX]
33	Asynchronous close in the layered storage system
34	Automatic readahead in the layered storage system
35	Whether to cache data in a compute node when reading from the layered storage system
36	Whether to cache data on first-layer storage when reading from second-layer storage
37	File storing the names of files that were unable to complete writing from first-layer storage to second-layer storage The metacharacter with % in front means the output file name format. It has the same meaning as a metacharacter that can be specified in --llio uncompleted-fileinfo-path in the pjsub command.
38	Whether to output LLIO performance information

No.	Description
39	Output destination file name of LLIO performance information The metacharacter with % in front means the output file name format. It has the same meaning as a metacharacter that can be specified in --llio perf-path in the pjsub command.
40	Group-related setting values
41	Group priority
42	Initial fair share value of the group
43	Fair share value recovery rate of the group
44	Upper limit and default values specified in pjsub command options
45	Upper limit and default values in a batch job
46	Limit on executable time per job
47	Minimum value for the executable time per job
48	Maximum value for the executable time per job
49	Limit on the number of requested nodes multiplied by time per job (*1)
50	Minimum value when the number of requested nodes is multiplied by the executable time per job (*1)
51	Maximum value when the number of requested nodes is multiplied by the executable time per job (*1)
52	Limit on the number of CPU cores multiplied by time per job (*1)
53	Limit on the number of CPU cores per job (*1)
54	Limit on the number of nodes per job
55	Limit on the memory used per node
56	Limit on the number of virtual nodes per job
57	Limit on the number of cores per a virtual node
58	Limit on the memory size allocated to a virtual node
59	Limit on the core file size per process
60	Limit on CPU time per process
61	Limit on the number of processes generated per process
62	Limit on the data segment size per process
63	Limit on the lock memory size per process
64	Limit on the message queue size per process
65	Limit on the number of file descriptors per process
66	Limit on the number of pending signals per process
67	Limit on the file size per process
68	Limit on the stack size per process
69	Limit on the virtual memory size per process
70	Limit on the number of custom resources used per resource unit or resource group (*2)
71	Limit on the number of custom resources used per node (limit on the number used by one job) (*2)
72	Limit on the number of custom resources used per node (limit on the number used per node by one job) (*2)
73	Limit on the number of custom resources used per node (limit on the number used per virtual node by one job) (*2)
74	Type and default value for a custom resource
75	Type that can be specified for a custom resource (*2)
76	Upper limit, lower limit, and default values for an interactive job

No.	Description
77	Upper limit, lower limit, and default values of job priority
78	Limit on the number of sub jobs of a bulk job
79	Upper limits, lower limits, and default values of parameters related to the layered storage system
80	Size of the shared temporary area Multiply this size by the number of allocated compute nodes, and the resulting value is the size of the shared temporary area available to one job.
81	Size of the node temporary area
82	Threshold value of whether or not to cache data when writing the data to first-layer storage When writing data to the first-layer storage, if the write size is equal to or less than this value, the data is not immediately written to the storage. Instead, it is temporarily cached in compute node cache.
83	Compute node cache size
84	Number of stripes per file when files are distributed across first-layer storage
85	Stripe size when files are distributed across first-layer storage
86	Execution privileges enabled or disabled
87	Privilege to execute the pjsub command
88	Privilege to specify the execution start time
89	Privilege to submit batch jobs
90	Privilege to submit interactive jobs
91	Privilege to submit normal jobs
92	Privilege to submit step jobs
93	Privilege to submit bulk jobs
94	Privilege to submit master-worker jobs
95	Privilege to submit jobs to which nodes are allocated
96	Privilege to submit jobs to which virtual nodes are allocated
97	Privilege to execute the pjstat command
98	Privilege to execute the pjdel command
99	Privilege to cancel prologue and epilogue scripts in pjdel
100	Privilege to specify the --no-history option that suppresses the output of the job information to the job history information when a job is deleted.
101	Privilege to execute the pjhold command
102	Privilege to cancel prologue and epilogue scripts in pjhold
103	Privilege to execute the pjrls command
104	Privilege to execute the pjwait command
105	Privilege to execute the pjalter command
106	Privilege to execute the pjsig command
107	Privilege to execute the pjacl command
108	Privilege to change the output destination of the standard output and standard error output of the mpiexec command in a batch job [FX]
109	Privilege to change the output destination of the standard output of the mpiexec command in a batch job [FX]
110	Privilege to change the output destination of the standard error output of the mpiexec command in a batch job [FX]

No.	Description
111	Privilege to change the output destination of the standard output and standard error output of the mpiexec command in an interactive job [FX]
112	Privilege to change the output destination of the standard output of the mpiexec command in an interactive job [FX]
113	Privilege to change the output destination of the standard error output of the mpiexec command in an interactive job [FX]
114	Privilege to submit jobs in torus mode [FX]
115	Privilege to submit jobs in mesh mode [FX]
116	Privilege to submit jobs in non-contiguous mode [FX]
117	Privilege to submit jobs in node allocation without strict and strict-io specified
118	Privilege to submit jobs in node allocation with strict specified
119	Privilege to submit jobs in node allocation with strict-io specified
120	Privilege to submit jobs with I/O-sharing mode (no-io-exclusive) specified [FX]
121	Privilege to submit jobs with I/O-exclusive mode (io-exclusive) specified [FX]
122	Privilege to specify a virtual node placement policy [PG]
123	Privilege to specify an execution mode policy [PG]
124	Privilege to submit jobs with the format specifying the executable time of a job (pjsub -L elapse=*limit*)
125	Privilege to submit jobs with the format specifying the minimum and maximum values for the executable time of a job (pjsub -L elapse=*min_limit*[-*max_limit*])
126	Privilege to specify whether to dynamically change the communication path when a link goes down for the Tofu interconnect used by a job (pjsub --net-route) [FX]
127	Privilege to use the command API
128	Permission for operated objects
129	Permission allowing execution by the group with the pjsub command
130	Target job of the pjstat command
131	Target job of the pjdel command
132	Target job of the pjhold command
133	Target job of the pjrls command
134	Target job of the pjwait command
135	Target job of the pjalter command
136	Target job of the pjsig command
137	Target job of the pjacl command
138	Target job of the pmerls command
139	Target job of the pmalter command
140	Users and groups permitted to display information with the pjshowrsc command
141	Limit values in a resource unit (each user)
142	Limit on concurrent acceptance for a batch job
143	Limit on concurrent acceptance for sub jobs of a bulk or step job, and normal job (batch job)
144	Limit on concurrent acceptance for sub jobs of a bulk job
145	Limit on concurrent acceptance for sub jobs of a step job
146	Limit on concurrent execution for a batch job
147	Limit on concurrent execution for sub jobs of a bulk job

No.	Description
148	Limit on concurrent node usage for a batch job
149	Limit on the number of CPU cores that can be used concurrently by batch jobs
150	Limit on concurrent acceptance for an interactive job
151	Limit on concurrent execution for an interactive job
152	Limit on concurrent node usage for an interactive job
153	Limit on the number of CPU cores that can be used concurrently by interactive jobs
154	Limit on the number of concurrently used custom resources per resource unit or resource group (*2)
155	Limit on the number of custom resources used concurrently by interactive jobs per resource unit or resource group (*2)
156	Limit values in a resource unit (total values in a group)
157	Limit values in a resource unit (total values for all users)
158	Limit values in a resource group (each user).
159	Limit values in a resource group (total values in a group)
160	Limit values in a resource group (total values for all users)

(*1) No option of the pjsub command corresponds to this item.
(*2) Actually, the names of defined custom resources are displayed at customrscname1 to customrscname4.

As shown above for the defined item "permit," "allow" indicates operation permitted, and "deny" indicates operation denied. The following table lists the meanings of the operated objects following "allow" or "deny."

Table 2.5 Notation of operated objects in the pjacl command

Notation	Description
own	The target is only one's own submitted jobs.
all	The target is the jobs of all users.
g(g1, g2, ...)	The target group names are g1, g2, and so on.
u(u1, u2, ...)	The target user names are u1, u2, and so on.

Note

- The Administrator may not permit by the job ACL function a user to execute the Job Operation Software commands described in this manual. Check the output results of the pjacl command.
  The user might not have a permission to execute the pjacl command.
- According to a job operation setting, node allocated jobs may be limited by the number of concurrently used nodes or the number of concurrently used CPU cores. Ask the administrator about which one of the above is set for job operations.
  If set to be limited by the number of concurrently used CPU cores, the number of CPU cores concurrently used by node allocated jobs is "number of mounted CPU cores per node x number of requested nodes."

Using the --limit option of the pjstat command, you can check the limit values on resources and current quotas for job submission by a user in the contents displayed by the pjacl command.

- Resource limit values and quotas in a resource unit

```
$ pjstat --limit --rscunit rscunit1
 System Resource Information:
 RSCUNIT: rscunit1
 USER: user1
  LIMIT-NAME                          LIMIT         ALLOC
  ru-accept                       unlimited             5     (1)
  ru-accept-allsubjob             unlimited             0     (2)
```

```
  ru-accept-bulksubjob             unlimited              0     (3)
  ru-accept-stepsubjob             unlimited              0     (4)
  ru-run-job                       unlimited              3     (5)
  ru-run-bulksubjob                unlimited              0     (6)
  ru-use-node                      unlimited           1011     (7)
  ru-interact-accept               unlimited              0     (8)
  ru-interact-run-job              unlimited              0     (9)
  ru-interact-use-node             unlimited              0     (10)
  ru-use-core                      unlimited             30     (11)
  ru-interact-use-core             unlimited              0     (12)
  ru-custom-customresourcename            10              1     (13)(*)
  ru-interact-custom-customresourcename    5              1     (14)(*)
 GROUP: group1
  LIMIT-NAME                           LIMIT          ALLOC
  ru-accept                        unlimited              5
  ru-accept-allsubjob              unlimited              0
  ru-accept-bulksubjob             unlimited              0
  ru-accept-stepsubjob             unlimited              0
  ru-run-job                       unlimited              3
  ru-run-bulksubjob                unlimited              0
  ru-use-node                      unlimited           1011
  ru-interact-accept               unlimited              0
  ru-interact-run-job              unlimited              0
  ru-interact-use-node             unlimited              0
  ru-use-core                      unlimited             30
  ru-interact-use-core             unlimited              0
ALL:
  LIMIT-NAME                           LIMIT          ALLOC
  ru-accept                        unlimited              5
  ru-accept-allsubjob              unlimited              0
  ru-accept-bulksubjob             unlimited              0
  ru-accept-stepsubjob             unlimited              0
  ru-run-job                       unlimited              3
  ru-run-bulksubjob                unlimited              0
  ru-use-node                      unlimited           1011
  ru-interact-accept               unlimited              0
  ru-interact-run-job              unlimited              0
  ru-interact-use-node             unlimited              0
  ru-use-core                      unlimited             30
  ru-interact-use-core             unlimited              0
```

(1) Limit on concurrent acceptance for a batch job
(2) Limit on concurrent acceptance for sub jobs of a bulk or step job, and normal jobs (batch job)
(3) Limit on concurrent acceptance for sub jobs of a bulk job
(4) Limit on concurrent acceptance for sub jobs of a step job
(5) Limit on concurrent execution for a batch job
(6) Limit on concurrent execution for sub jobs of a bulk job
(7) Limit on concurrent node usage for batch job
(8) Limit on concurrent acceptance for an interactive job
(9) Limit on concurrent execution for an interactive job
(10) Limit on concurrent node usage for an interactive job
(11) Limit on the number of CPU cores that can be used concurrently by batch jobs
(12) Limit on the number of CPU cores that can be used concurrently by interactive jobs
(13) Limit on the number of concurrently used custom resources (*)
(14) Limit on the number of custom resources used concurrently by interactive jobs (*)
(*) Actually, the name of a defined custom resource is displayed at *customresourcename* in *ru-custom-customresourcename* and *ru-interact-custom-customresourcename*.

- Resource limit values and quotas in a resource group

```
$ pjstat --limit --rscunit rscunit1 --rscgrp grp1
 System Resource Information:
```

```
RSCUNIT: rscunit1
RSCGRP : grp1
USER: user1
 LIMIT-NAME                          LIMIT         ALLOC
 rg-accept                         unlimited           5   (1)
 rg-accept-allsubjob               unlimited           0   (2)
 rg-accept-bulksubjob              unlimited           0   (3)
 rg-accept-stepsubjob              unlimited           0   (4)
 rg-run-job                        unlimited           3   (5)
 rg-run-bulksubjob                 unlimited           0   (6)
 rg-use-node                       unlimited        1011   (7)
 rg-interact-accept                unlimited           0   (8)
 rg-interact-run-job               unlimited           0   (9)
 rg-interact-use-node              unlimited           0   (10)
 rg-use-core                       unlimited          30   (11)
 rg-interact-use-core              unlimited           0   (12)
 rg-custom-customresourcename              5           1   (13)(*)
 rg-interact-custom-customresourcename     3           1   (14)(*)
GROUP: group1
 LIMIT-NAME                          LIMIT         ALLOC
 rg-accept                         unlimited           5
 rg-accept-allsubjob               unlimited           0
 rg-accept-bulksubjob              unlimited           0
 rg-accept-stepsubjob              unlimited           0
 rg-run-job                        unlimited           3
 rg-run-bulksubjob                 unlimited           0
 rg-use-node                       unlimited        1011
 rg-interact-accept                unlimited           0
 rg-interact-run-job               unlimited           0
 rg-interact-use-node              unlimited           0
 rg-use-core                       unlimited          30
 rg-interact-use-core              unlimited           0
ALL:
 LIMIT-NAME                          LIMIT         ALLOC
 rg-accept                         unlimited           5
 rg-accept-allsubjob               unlimited           0
 rg-accept-bulksubjob              unlimited           0
 rg-accept-stepsubjob              unlimited           0
 rg-run-job                        unlimited           3
 rg-run-bulksubjob                 unlimited           0
 rg-use-node                       unlimited        1011
 rg-interact-accept                unlimited           0
 rg-interact-run-job               unlimited           0
 rg-interact-use-node              unlimited           0
 rg-use-core                       unlimited          30
 rg-interact-use-core              unlimited           0
```

(1) Limit on concurrent acceptance for a batch job
(2) Limit on concurrent acceptance for sub jobs of a bulk or step job, and normal jobs (batch job)
(3) Limit on concurrent acceptance for sub jobs of a bulk job
(4) Limit on concurrent acceptance for sub jobs of a step job
(5) Limit on concurrent execution for a batch job
(6) Limit on concurrent execution for sub jobs of a bulk job
(7) Limit on concurrent node usage for batch job
(8) Limit on concurrent acceptance for an interactive job
(9) Limit on concurrent execution for an interactive job
(10) Limit on concurrent node usage for an interactive job
(11) Limit on the number of CPU cores that can be used concurrently by batch jobs
(12) Limit on the number of CPU cores that can be used concurrently by interactive jobs
(13) Limit on the number of concurrently used custom resources (*)
(14) Limit on the number of custom resources used concurrently by interactive jobs (*)

(*) Actually, the name of a defined custom resource is displayed at *customresourcename* in *rg-custom-customresourcename* and *rg-interact-custom-customresourcename*.

The following table lists the items displayed by the --limit option of the pjstat command and their meanings.

Item	Description
LIMIT-NAME	Limit value name
RSCUNIT	Resource unit name
GROUP	Group name in the operating system
LIMIT	Limit value
ALLOC	Current allocation value

2.2.3 Checking resources

Use the pjshowrsc command to check the usage of resources in the system.

You can use the --rscunit (or --ru) option to display the usage of resources of the resource unit executing the job. You can use the -E option to display a lower layer.

Information

Short option --ru is prepared for the --rscunit option. You can specify these options in either format.

Below, the option to specify resource units is called the --rscunit (--ru) option.

```
$ pjshowrsc --rscunit -E
[ CLST: clst ]
[ RSCUNIT: unit1 ]
    NODEID     CPU                    MEM
             TOTAL    FREE  ALLOC   TOTAL    FREE   ALLOC
0x01010010       8       0      8    14Gi       0    14Gi
0x01010011       8       0      8    14Gi       0    14Gi
0x01010012       8       0      8    14Gi       0    14Gi
0x01010013       8       0      8    14Gi       0    14Gi
0x01010014       8       8      0    14Gi    14Gi       0
```

The meanings of the displayed items are as follows.

Item		Meaning
CPU	TOTAL	Number of CPU cores mounted on nodes
	FREE	Number of CPU cores not allocated to jobs
	ALLOC	Number of CPU cores already allocated to jobs
MEM	TOTAL	Amount of memory mounted on nodes (byte)
	FREE	Amount of memory not allocated to jobs (byte)
	ALLOC	Amount of memory already allocated to jobs (byte)

Note

In environments in which a local file system does not exist, all values of item LFS of the pjshowrsc command are "-".

You can display the usage of a custom resource by specifying the --custom-resource option.

```
$ pjshowrsc --rscunit unit1 --custom-resource
[ CLST: clst ]
RSCUNIT          NODE
                 TOTAL   FREE  ALLOC
unit1               96     96      0

CUSTOM RESOURCE                            (*) Usage of custom resource per resource unit
RSCNAME                             TOTAL      FREE     ALLOC
customrscname1                    1000000    300000    700000
customrscname2/type             unlimited unlimited         2

CUSTOM RESOURCE(PER NODE)                  (*) Usage of custom resource per node
RSCNAME                             TOTAL      FREE     ALLOC     NODE
customrscname3                         10        10         0       10
```

(*1) The names of defined custom resources are displayed at *customrscname1* and *customrscname3*. Also, a custom resource name/type is displayed at *customrscname2/type*, which is a custom resource defined with a *type*.
(*2) The information on custom resources per node (CUSTOM RESOURCE (PER NODE)) displays the sum of custom resources of compute nodes in a resource unit.

Information

The short option -C is prepared for the --custom-resource option. You can specify the option in either format.

The meanings of the displayed items are as follows.

Item	Meaning
RSCNAME	Custom resource name For a custom resource defined with a type, the custom resource name/type is displayed in the *customrscname/type* format.
TOTAL	Total quantity of custom resources For a custom resource defined with a type, unlimited is displayed.
FREE	Total quantity of custom resources not allocated to jobs For a custom resource defined with a type, unlimited is displayed.
ALLOC	Total quantity of custom resources already allocated to jobs (see "2.3.2.4 Specifying a custom resource") The calculation for a custom resource defined with a type assumes the specified numerical number of custom resources is 1.
NODE	Number of nodes defined for a custom resource This item is output only for custom resources per node.

The pjshowrsc command displays the resource units for which the user has privileges. However, to check the default resource group for the user, uses the --rsc option of the pjstat command. (See "2.2.1 Checking resource units and resource groups.")

To check the usage of the resources of a resource group in a resource unit, specify the --rscgrp (or --rg) option.

Information

Short option --rg is prepared for the --rscgrp option. You can specify these options in either format.

Below, the option to specify resource units is called the --rscgrp (--rg) option.

- Displaying a summary for each resource group
  Specify the --rscgrp (--rg) option without arguments to display a summary of all resource groups.

```
$ pjshowrsc --rscunit unit1 --rscgrp
[ CLST: clst ]
```

```
[ RSCUNIT: unit1 ]
RSCGRP           NODE
                 TOTAL   FREE  ALLOC
group1              36     24     12
group2              36     24     12
group3              36     24     12
```

- Displaying a summary for a specific resource group
  To check a summary of a specific resource group, specify the name of the resource group as an argument of the --rscgrp (--rg) option.

```
$ pjshowrsc --rscunit unit1 --rscgrp group2
[ CLST: clst ]
[ RSCUNIT: unit1 ]
RSCGRP           NODE
                 TOTAL   FREE  ALLOC
group2              36     24     12
```

- Displaying details on the specified resource group
  Specify the -l option to display details on the resources of the specified resource group.

```
$ pjshowrsc --rscunit unit1 --rscgrp group2 -l
[ CLST: clst ]
[ RSCUNIT: unit1 ]
[ RSCGRP: group2 ]
     RSC    TOTAL    FREE   ALLOC
    node       36      33       3
     cpu     1152    1056      96
     mem    941Gi   856Gi     85Gi
```

- Displaying a list of the resources in a resource group
  Specify the -E option to display a list of the resources in the specified resource group by node.

```
$ pjshowrsc --rscunit unit1 --rscgrp group1 -E
[ CLST: clst ]
[ RSCUNIT: unit1 ]
[ RSCGRP: group1 ]
    NODEID     CPU                    MEM
             TOTAL    FREE  ALLOC   TOTAL    FREE   ALLOC
0xFF030001      32      32      0    29Gi    29Gi       0
0xFF030002      32      32      0    29Gi    29Gi       0
0xFF030003      32      32      0    29Gi    29Gi       0
...
```

- Displaying details on the resources in a resource group
  Specify the -v option to display detailed information on the resources in the specified resource group by node.

```
$ pjshowrsc --rscunit unit1 --rscgrp group1 -v
[ CLST: clst ]
[ RSCUNIT: unit1 ]
[ RSCGRP: group1 ]
[ NODE: 0xFF030001 ]
     RSC    TOTAL    FREE   ALLOC
     cpu       32      32       0
     mem     29Gi    29Gi       0

[ NODE: 0xFF030002 ]
     RSC    TOTAL    FREE   ALLOC
     cpu       32      32       0
     mem     29Gi    29Gi       0

[ NODE: 0xFF030003 ]
     RSC    TOTAL    FREE   ALLOC
     cpu       32      32       0
```

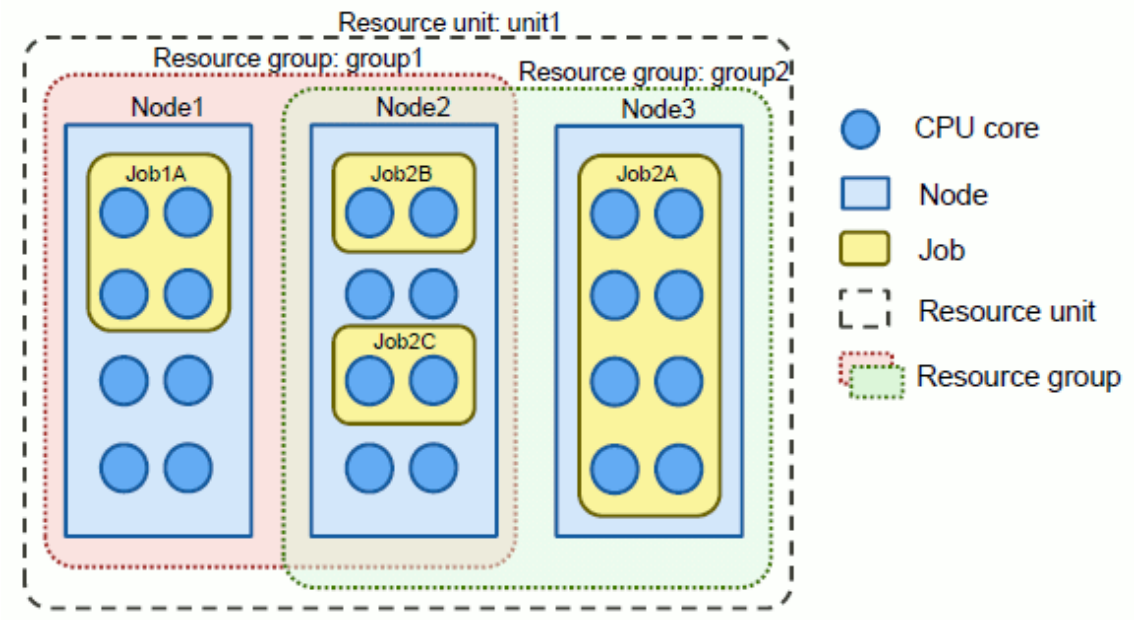
```
     mem     29Gi    29Gi       0
...
```

The following table lists the meanings of the items displayed when the --rscgrp (--rg) option is specified.

Item	Meaning
RSC	Type of resource (cpu: number of CPU cores, mem: amount of memory)
TOTAL	Maximum amount of resources of the resource group (Note)
FREE	The displayed value varies depending on whether the --exclusive option is specified. - When the --exclusive option is not specified: The amount of free resources available in the relevant resource group is displayed. - When the --exclusive option is specified: "-" is displayed. The amount of free resources is not displayed. Since the resource group specified in --rscgrp may share resources with another resource group, the amount of the resource group's own free resources cannot be calculated.
ALLOC	The displayed value varies depending on whether the --exclusive option is specified. - When the --exclusive option is not specified: Amount of resources used by the relevant resource group and the resource group with which it shares resources - When the --exclusive option is specified: Amount of resources used only by the relevant resource group

(Note)
In the resource group of FX server, all the resources in the resource unit are displayed when defining it in the method of specifying the resource that the manager allocates to the resource group at the number of nodes or the rate. However, when the resource group that sets the execution mode policy to "simplex" in the resource unit exists, the resource of this resource group is excluded and displayed. Confirm the content of the definition of the resource of the resource group to the manager.

The --exclusive option is used to check the amount of resources used only by the relevant resource group when it shares resources with another resource group.
The following describes, as an example, differences in display depending on whether the --exclusive option is specified, under the resource use conditions shown below.



Resource unit: unit1	Consists of 3 nodes (nodes 1, 2, 3), each of which has 8 cores and 30 GiB of memory.

Resource group: group1	Consists of nodes 1 and 2. Node 2 is shared with the resource group group2.
Resource group: group2	Consists of nodes 2 and 3. Node 2 is shared with the resource group group1.
Job 1A	Executed on the resource group group1. It uses 4 CPU cores and 20 GiB of memory on node 1.
Job 2A	Executed on the resource group group2. It uses 8 CPU cores and 20 GiB of memory on node 3.
Job 2B	Executed on the resource group group2. It uses 2 CPU cores and 10 GiB of memory on node 2.
Job 2C	Executed on the resource group group2. It uses 2 CPU cores and 10 GiB of memory on node 2.

In this case, the pjshowrsc command displays the status of resources in each resource group, showing the following results.

- When the --exclusive option is not specified
  The item ALLOC of the resource group group1 includes the amount of resources used by the resource group group2, with which resources are shared.

```
$ pjshowrsc --rscunit unit1 --rscgrp -l
[ CLST: clst000 ]
[ RSCUNIT: unit1 ]
[ RSCGRP: group1 ]
     RSC    TOTAL    FREE   ALLOC
    node        2       0       2
     cpu       16       8       8
     mem     60Gi     20Gi    40Gi
 [ RSCGRP: group2 ]
     RSC    TOTAL    FREE   ALLOC
    node        2       0       2
     cpu       16       4      12
     mem     60Gi     20Gi    40Gi
```

- When the --exclusive option is specified
  The item ALLOC displays only the amount of resources used by each resource group. In addition, when the --exclusive option is specified, the item FREE displays "-".

```
$ pjshowrsc --rscunit unit1 --rscgrp -l --exclusive
[ CLST: clst ]
[ RSCUNIT: unit1 ]
[ RSCGRP: group1 ]
     RSC    TOTAL    FREE   ALLOC
    node        2       -       1
     cpu       16       -       4
     mem     60Gi       -    20Gi
[ RSCGRP: group2 ]
     RSC    TOTAL    FREE   ALLOC
    node        2       -       2
     cpu       16       -      12
     mem     60Gi       -    40Gi
```

2.3 Submitting a Job

This section describes how to submit a job.

2.3.1 Basic methods of submitting a job

Use the pjsub command to submit a job.

The following is a simple example of submitting the job script job.sh.

```
$ pjsub job.sh
```

When submitting a job, you can use options of the pjsub command to specify the amounts of resources (number of nodes, amount of memory, etc.) to allocate to the job. The following examples of submitting a job specify the number of nodes or virtual nodes to allocate to the job.

```
[Example of allocating 8 nodes to the job]
$ pjsub -L "node=8" job.sh

[Example of allocating 8 virtual nodes to the job]
$ pjsub -L "vnode=8" job.sh
```

The specifiable amounts of resources are in the allowable range defined by the job ACL function. You can specify an amount in that range. If the amount of resources is not specified, the value that is set by the job ACL function is applied. For details on how to check the value, see "2.2.2 Checking restriction information." For the details of the options which can be specified on submitting a job, see "2.3.2 Options at the job submission time" and subsequent explanation, and the man page of the pjsub.

Note

- On some systems, a single cluster may contain multiple resource units or groups that differ in terms of job operation policy, resource quantity, and/or compute node model.
  If necessary on such systems, specify the resource unit or group to which to submit jobs. (See "2.3.2.1 Specifying resources.")
  You can use the pjacl command to check the default resource unit or group to which to submit jobs. (See "2.2.1 Checking resource units and resource groups.")
- The pjsub command ignores the options of the FX server for the PRIMERGY servers. When the options of the PRIMERGY server are specified to the pjsub command for the FX servers, an error occurs.

Information

You can write the options of the pjsub command not only on the command line but also in a job script.
However, the pjsub command options specified on the command line have priority over those specified in a job script.

If no job script is specified, the job contents are read from the standard input of the pjsub command.

```
$ pjsub
#!/bin/sh          <- The user enters the job contents to the standard input of pjsub.
...
<Ctrl+d>           <- The user closes the standard input with the <Ctrl+d> keys.
```

The following message appears when the job is accepted normally.

```
[INFO] PJM 0000 pjsub Job jobid submitted.
```

jobid is set as the job ID for the submitted job.

Also, "job name" is set for the job. The default is the job script name. If you want to change it, you can specify a name in the -N or --name option of the pjsub command when submitting a job.

2.3.2 Options at the job submission time

This section describes the pjsub command options for specifying resource allocation for a job and job operations.

2.3.2.1 Specifying resources

You can specify the resources to be allocated to a job by using the -L option or --rsc-list option of the pjsub command.

```
{-L | --rsc-list } item=value
```

You can specify the following items and values.

Table 2.6 Resource specification formats

Format	Description
For FX server: node=*shape*[:strict][:*node_allocation*] For PRIMERGY server: node=*N*	Number and shape of nodes allocated to a job, and how to allocate nodes to a job (for a node allocated job) For details, see "2.3.2.2 Specifying a node resource."
vnode=*num*[*CPU cores or memory size*]	Number of virtual nodes, number of CPU cores, and amount of memory allocated to a job (for a virtual node allocated job) For details, see "2.3.2.2 Specifying a node resource." The FX server does not support allocating virtual nodes.
elapse=*limit* For the FX server, the following formats are also available: elapse=*min_limit*- elapse=*min_limit*-*max_limit*	Elapsed time limit value for a job (executable time) *limit* specifies the upper limit of the job elapsed time. For jobs that are allocated FX servers, you can also specify an upper limit for the elapsed time as a range from the minimum value *min_limit* to the maximum value *max_limit*. If the node has free space when the job elapsed time reaches the minimum value *min_limit*, job execution continues until the elapsed time reaches the maximum value *max_limit*. Specify a value of at least 1 second for the minimum value *min_limit*, and a value of at least 2 seconds for the maximum value *max_limit*. For details on how to specify the elapsed time limit value, see "2.3.2.3 Specifying the elapsed time limit value for a job."
node-mem=*limit*	Upper limit on memory usage per node (for a node allocated job) The minimum value is 1 MiB (mebibyte = 2^{20} bytes).
rscunit=*name* ru=*name*	Resource unit name for submitting a job *name* is a character string with up to 63 characters.
rscgrp=*name* rg=*name*	Resource group name for submitting a job *name* is a character string with up to 63 characters.
proc-core=*limit*	Maximum core file size for each process per job
proc-cpu=*limit*	Maximum CPU time for each process per job The minimum value is 1 second.
proc-crproc=*limit*	Maximum number of processes that can be generated with the real user ID on the node executing a process
proc-data=*limit*	Maximum data segment size for each process per job
proc-lockm=*limit*	Maximum lock memory size for each process per job
proc-msgq=*limit*	Maximum POSIX message queue size that can be reserved with the real user ID on the node executing a process
proc-openfd=*limit*	Maximum number of file descriptors for each process per job
proc-psig=*limit*	Maximum number of pending signals with the real user ID on the node executing a process
proc-filesz=*limit*	Maximum file size for each process per job
proc-stack=*limit*	Maximum stack size for each process per job
proc-vmem=*limit*	Maximum virtual memory size for each process per job
CustomResourceName=*Value*	Requested quantity or requested type of a custom resource *CustomResourceName* is the name of a custom resource, and *Value* is the numerical number or the type of the custom resource.

Format	Description
	"= *Value*" can be omitted. For details, see "2.3.2.4 Specifying a custom resource."

Specify *limit* in the expression format for time shown as follows.

Table 2.7 Expression formats for amounts of resources

Amount	Expression format
Time	You can specify a time in seconds or in the MM:SS, HH:MM:SS, or HHhMMmSSs format. You can also use the uppercase letters of h, m, and s. Unless otherwise noted, the specifiable times are in a range of 1 to 2147483647 seconds. For no limit, specify unlimited. Example: elapse=100 (100 seconds) elapse=1:40 (1 minute 40 seconds) elapse=6:10:30 (6 hours 10 minutes 30 seconds) elapse=10h30m50s (10 hours 30 minutes 50 seconds) elapse=300m (300 minutes) elapse=unlimited (No limit)
Amount of memory (bytes)	Express the value as a power of two (unit). The units Ki (kibi = 2^{10}), Mi (mebi = 2^{20}), Gi (gibi = 2^{30}), Ti (tebi = 2^{40}), and Pi (pebi =2^{50}) are powers of two. If omitted, Mi is assumed specified. Do not enter any space between the numerical value and the unit representing a power of two. Unless otherwise noted, the specifiable values are in a range of 0 bytes to 2147483647 MiB. For no limit, specify unlimited. Example: node-mem=256Mi (256MiB = 256*2^{20} bytes) node-mem=256 (Same as above) node-mem=10Gi (10GiB = 10*2^{30} bytes) node-mem=unlimited (No limit)
Disk capacity File size (bytes)	Express the value as a power of 10 (unit). The units K (kilo = 10^3), M (mega = 10^6), G (giga = 10^9), T (tera = 10^{12}), and P (peta =10^{15}) are powers of 10. If omitted, M is assumed specified. Do not enter any space between the numerical value and the unit representing a power of 10. Unless otherwise noted, the specifiable values are in a range of 0 bytes to 2147483647 MB. For no limit, specify unlimited. Example: node-quota=10M (10MB = 10*10^6 bytes) node-quota=10 (Same as above) node-quota=10G (10GB = 10*10^9 bytes) node-quota=unlimited (No limit)
Numerical value	For the amount of any resource other than the above, specify only a numerical value. Unless otherwise noted, the specifiable values are in a range of 0 to 2147483647. For no limit, specify unlimited.

These values can neither exceed the upper limit values nor fall below the lower limit values defined by the job ACL function.
These values cannot exceed the upper limit values defined by the job ACL function. For any option not specified, the corresponding default value defined by the job ACL function is applied. To check the values defined by the job ACL function, see "2.2.2 Checking restriction information."

For details on how a job behaves when it uses more resources than the specified upper limit, see "2.3.2.5 Job operation when a job exceeds an upper limit on amount of resources."

The following examples show the specification of job resources.

- Example of allocation of 64 nodes to a job

```
$ pjsub -L "node=64" job.sh
```

  The job is executed on the specified number of nodes. The job cannot use more than the specified number of nodes. For example, suppose an MPI program requires multiple nodes. If the MPI program requires more nodes than the allocated number of nodes, the program ends with an error.

- Example of setting 86400 seconds as the executable time of a job

```
$ pjsub -L "elapse=86400" job.sh
```

  If the running job exceeds the specified job executable time, the job is forcibly terminated.

- Example of setting a 256 MiB limit on the memory used for each node of a job.

```
$ pjsub -L "node-mem=256Mi" job.sh
```

  The job is executed within this limit on the memory used for each node. Moreover, not even with a request can the limit on the memory used per node be exceeded.

The following example shows a combination of the above-described options.

This example submits the MPI program prog with the following limit values: number of allocated nodes (2), executable time (86400 seconds), and upper limit on the memory used per node (256 MiB). Also, the job script job.sh, not the arguments of the pjsub command, specifies the limit values.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=2"                                                    (1)
#PJM -L "elapse=86400"                                              (2)
#PJM -L "node-mem=256Mi"                                            (3)
...
mpiexec ./prog                                                      (4)
$ pjsub job.sh
```

(1) Number of allocated nodes: 2
(2) Executable time: 86400 seconds
(3) Upper limit on the memory used per node: 256MiB
(4) mpiexec command executing the MPI program prog

For details of mpiexec command in above example, see "MPI User's Guide", which is a Development Studio manual.

When submitting a job, you can specify not only the resources to allocate to the job but also first-layer storage-related parameters for job input/output. For details, see "2.3.3 Specifying LLIO-related parameters [FX]."

2.3.2.2 Specifying a node resource

Use the node or vnode parameter of the -L (--rsc-list) option of the pjsub command to specify the node resources to allocate to a job. Use the node parameter for allocation in units of nodes and the vnode parameter for allocation in units of virtual nodes.

If neither of the node and vnode parameters is specified, the relevant setting of the job ACL function determines whether to allocate nodes or virtual nodes. Execute the pjacl command to check the setting value of the item default alloc granularity.

[Allocation in units of nodes]

The method for specifying the node parameter to allocate node resources in units of nodes varies depending on the model of the compute node.

- To allocate FX servers

  When submitting a job, the user can select a one-dimensional, two-dimensional, or three-dimensional space as the space for placing the job. The user specifies the node shape in this logical space. For details on node shapes, see "1.6 Node Resource Allocation."

  Use the node parameter of the -L or --rsc-list option of the pjsub command to specify the node shape, which must be able to store all existing processes during job execution.

```
{-L | --rsc-list} "node=shape[:strict|:strict-io][:torus|:mesh|:noncont]"
```

Table 2.8 Node shape specification formats [FX]

Format	Description
shape[:strict|:strict-io]	Specify the size along each axis in shape, according to the dimensions of the node shape, which is X, XxY, or XxYxZ. If the node shape is three-dimensional, you can specify strict, such as in XxYxZ:strict. Normally, the allocated node shape as specified is automatically rotated to fit in the available node space. However, if strict is specified, the allocated node shape is not rotated. To express one-dimensional and two-dimensional shapes in the same way, use the strict specification in a three-dimensional shape expression. If the node parameter is not specified, the allocated node shape conforms to the default value defined by the job ACL function. Execute the pjacl command to check the default value of the item "(node=)" in "pjsub option parameters." When using the strict-io specification with compute nodes that are in the FX server, you can specify that the locations of I/O nodes always remain the same with regard to their Tofu coordinates. For details on the strict-io specification, see "2.3.2.11 Allocating resources in consideration of I/O nodes [FX]."
:torus or :mesh or :noncont	For a node allocated job, you can specify a node placement method (torus, mesh, or non-contiguous mode) in Tofu coordinates. The string "torus" means torus mode, in which computer resources are allocated to jobs in Tofu (12 nodes) units. The string "mesh" means mesh mode, in which computer resources are allocated to jobs in units of nodes. The string "noncont" means non-contiguous mode, in which computer resources are allocated to jobs in units of nodes. For details on each method of allocating nodes, see "1.6.3.1 Allocation in units of nodes for FX servers." When omitted, the default value defined with the job ACL function is followed. Execute the pjacl command to check the value of the item default allocation mode in "defines."

The following example allocates, in torus mode, a three-dimensional node shape (four X-, three Y-, and two Z-axis nodes) to a job that executes the MPI program prog_A.

```
$ cat job.sh                                                    (1)
#!/bin/sh
...
mpiexec ./prog_A
$ pjsub -L "node=4x3x2:torus" job.sh                            (2)
```

(1) Job script job.sh that executes the MPI program prog_A
(2) Submission of the job with a 4x3x2 node shape and torus mode specified

Note

- The specified node shape cannot exceed the maximum dimensions of the system. If it exceeds the maximum dimensions, the job submission will failed. For details on how to find out the maximum dimensions, see "2.2.1 Checking resource units and resource

groups."
The node shape may not fit within the maximum dimensions as is but may fit when rotated. If so, the node shape is automatically rotated. The user does not need to specify the rotated shape.

- Each of node allocation methods ":mesh," ":torus," and ":noncont" can be specified only when the corresponding function item in the job ACL (execute pjsub(torus), execute pjsub(mesh), and execute pjsub(noncont), respectively) is set to "enable."
- If the parameters node=*shape*, which allocates nodes, and vnode=*num* (described below), which allocates virtual nodes, are specified concurrently, a job submission error occurs.

- To allocate PRIMERGY servers
  To allocate PRIMERGY servers to a job, specify the number of nodes.

```
{-L | --rsc-list} "node=num
```

Table 2.9 Syntax for specifying the number of nodes [PG]

Parameter	Description
node=*num*	Specifies the number of nodes to allocate. If the node parameter is not specified, the number of nodes conforms to the default value defined by the job ACL function. Execute the pjacl command to check the default value of the item "(node=)" in "pjsub option parameters." When the vnode parameter is also specified, node resources are allocated in units of virtual nodes to jobs.

The following example allocates eight nodes to a job.

```
$ pjsub -L "node=8" job.sh
```

[Allocation in units of virtual nodes]

Specify the vnode parameter, which allocates node resources in units of virtual nodes, as shown below.

```
{-L | --rsc-list} "vnode=[num][,vnode-core=num][,{core-mem=size|vnode-mem=size}][,node=num]"
or
{-L | --rsc-list} "vnode=[num][([core=num][;core-mem=size|mem=size])][,node=num]"
```

Table 2.10 Parameters used for virtual node specification

Parameter	Description
[*num*]	Number of virtual nodes to allocate. If *num* is omitted or if the vnode parameter is not specified, the number of virtual nodes to be allocated conforms to the default value defined by the job ACL function. Execute the pjacl command to check the default value of the item "(vnode=)" in "pjsub option parameters."
vnode-core=*num* core=*num*	Number of CPU cores per virtual node If omitted, the value conforms to the relevant setting of the job ACL function. Execute the pjacl command to check the default value of the item "(vnode-core=)" in "pjsub option parameters."
vnode-mem=*size* mem=*size*	Amount of memory per virtual node. This parameter is used exclusively with the core-mem parameter. If omitted, the value conforms to the relevant setting of the job ACL function. Execute the pjacl command to check the default value of the item "(vnode-mem=) " in "pjsub option parameters."
core-mem=*size*	Amount of memory per CPU core. This parameter is used exclusively with the mem and vnode-mem parameter. If omitted, the value is determined by the number of CPU cores per virtual node and the amount of memory per virtual node.
node=*num* [PG]	Available only when the virtual node placement policy for PRIMERGY servers is UNPACK (-P "vn-policy=unpack"). This parameter specifies the number of nodes to allocate to the job. When the virtual node placement policy other than UNPACK is specified, this parameter is ignored. If this parameter is specified, the specified number of nodes is always required. For conceptual diagrams of allocation, see "Figure 2.15 UNPACK and specification of the number of nodes (1)" and "Figure 2.16 UNPACK and specification of the number of nodes (2)" in "2.3.5.1 Virtual node

Parameter	Description
	placement policy." If this parameter is omitted, all available nodes are allocated.

For the memory amount expression format, see "Table 2.7 Expression formats for amounts of resources."

The following example shows the number of virtual nodes is 1, the number of CPU cores in a virtual node is 5, and the memory amount per virtual node is 30MiB.

```
$ pjsub -L "vnode=1,vnode-core=5,vnode-mem=30Mi" job.sh
or
$ pjsub -L "vnode=1(core=5;mem=30Mi)" job.sh
```

Note

- The specification of "unlimited" for the core-mem, vnode-mem or mem parameter means that the job management function places no memory limitation on the job. The job can use as much memory as permitted by the OS. However, if another job begins to run on the same node, the job may run short of memory.
  If you submit a job whose limit value on the memory usage is "unlimited," use either of the following methods so that the job can occupy the node to prevent the memory acquisition failure due to resource competition with other jobs on the same node.
  - Submit a node allocated job (specify the node parameter).
  - If you submit a virtual node allocated job to the PRIMERGY servers, select the execution mode policy SIMPLEX for the job (-P exec-policy=simplex).
- The FX server does not support the virtual node allocated job.

See

In addition to the above-described amount of node resources to be allocated, the node allocation concept (node selection policy) can be specified for PRIMERGY servers. For details, see "2.3.5 Specifying a node selection policy [PG]."

2.3.2.3 Specifying the elapsed time limit value for a job

When submitting a job, you can use the following formats to specify the elapsed time limit value for the job (executable time).

- Specifying an elapsed time limit value

  ```
  {-L | --rsc-list} elapse=limit
  ```

  The job is forcibly terminated when the job elapsed time reaches *limit*.

- Specifying an elapsed time limit value as a range [FX]

  ```
  {-L | --rsc-list} elapse=min_limit-                (*) Hyphen required after min_limit
  {-L | --rsc-list} elapse=min_limit-max_limit
  ```

  After the elapsed time reaches *min_limit*, job execution can continue only until the elapsed time reaches *max_limit*. However, even when the elapsed time is less than *max_limit*, the job is forcibly terminated in the following situation:

  - the job is executed after the elapsed time passed *min_limit*;
  - the node is needed for a subsequent job, or the node enters a period reserved by deadline scheduling.

  To not restrict the longest time that execution can continue for, specify unlimited in *max_limit* (e.g., elapse=600-unlimited). If *max_limit* is not specified (e.g., elapse=600-), the value that is set by the job ACL function is applied to *max_limit*.

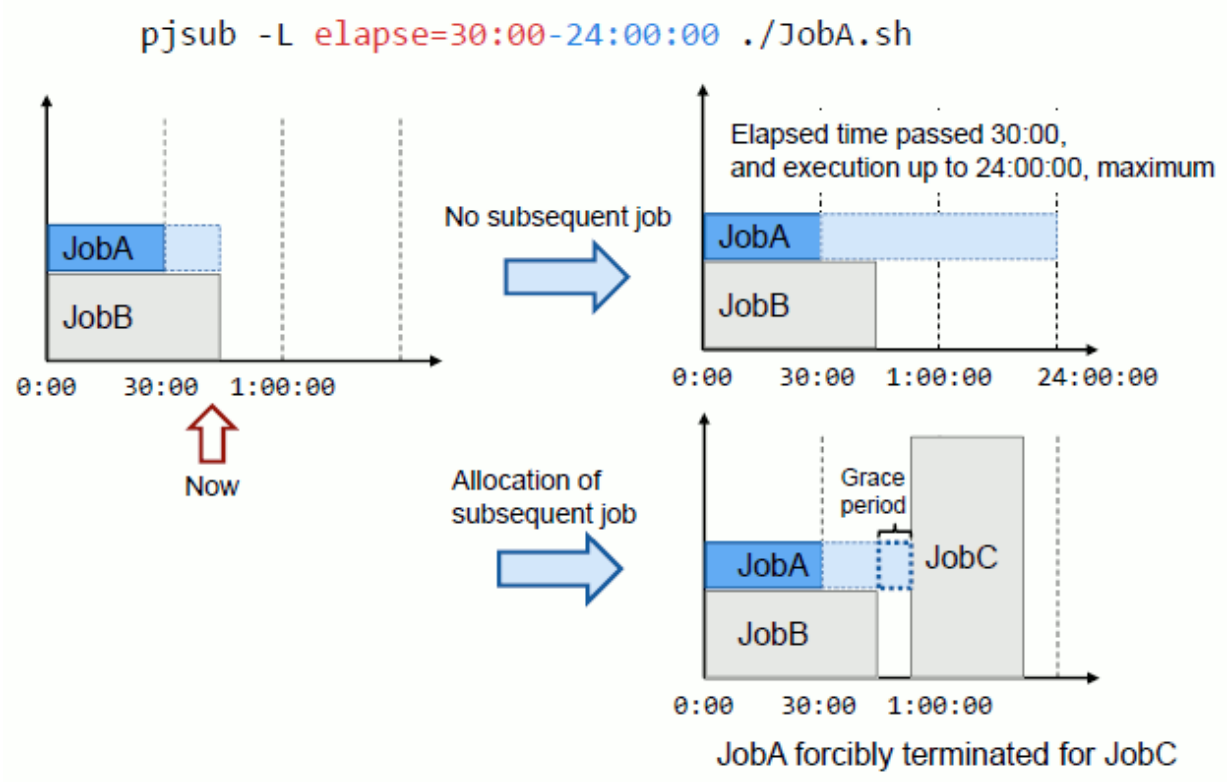
Information

Deadline scheduling is a function that prevents jobs from being allocated a node during a specific period for maintenance or other purposes. The administrator sets this function.

Note

If a setting by the administrator adds the execution time of the prologue and epilogue processes to the job elapsed time, the job may be forcibly terminated before the elapsed time reaches *min_limit*.

Figure 2.3 Specifying a range for the elapsed time limit value for a job



If a job reaches the elapsed time limit value and is then forcibly terminated, a signal will be sent before a grace period is set to allow end processing by the job. If necessary, capture the signal in the job to perform the processing.

- When specified with an upper limit value for the elapsed time (elapse=*limit*)

  The SIGXCPU signal is sent to the job 10 seconds before the forced termination.

- When specified with a range for the elapsed time (elapse=*min_limit*-*max_limit*)

  - If the job will be forcibly terminated before the elapsed time reaches *max_limit*, the SIGTERM signal is sent to the job 10 seconds before the termination. The administrator can change the length of the grace period in this specification format.

  - If the elapsed time has passed *max_limit*, the SIGXCPU signal is sent to the job 10 seconds before the forced termination.

Note

- Using the job ACL function, the administrator may have restricted use of the specification format of the elapsed time limit value. Check the "pjsub (fixed elapsed time)" and "pjsub (adaptive elapsed time)" parameters of the execute item with the pjacl command.

- If you do not specify the elapse parameter when submitting a job, the job ACL function applies either the elapse=*limit* format or the elapse=*min_limit-max_limit* format to the elapsed time limit value. Check the item "default elapsed time mode" item with the pjacl command.

2.3.2.4 Specifying a custom resource

To allocate a custom resource to a job, specify it in the following format with the -L option of the pjsub command.

```
{-L | --rsc-list} CustomResourceName[=Value]
```

CustomResourceName: Custom resource name

Value: Numerical number or type of allocated custom resources

You can specify up to 64 custom resources in a single job.

You can omit specifying *Value*.

The following table shows that the requested quantity or requested type that is adopted for a custom resource varies depending on the type of specified custom resources and the method of specifying parameters. (The types of specified custom resources are custom resources defined with a numerical value and custom resources defined with a type.)

Table 2.11 Requested quantity or requested type that is adopted for a custom resource for each method of specifying parameters

Type of specified custom resource	How to specify parameter		
	Specify *CustomResourceName=Value*	Specify only *CustomResourceName* (*Value* omitted)	Specify nothing
Specified custom resource defined with numerical value	The requested quantity is the numeric number of resources specified in *Value.*	The requested quantity is the default value defined by the job ACL function. If the default value is not defined, the requested quantity is the lower limit value.	The requested quantity is the default value defined by the job ACL function. If the default value is not defined, it is treated as if the pertinent custom resource is not used.
Specified custom resource defined with type	The requested type is the type of resources specified in *Value.*	The requested type is the default type defined by the job ACL function. If the default value is not defined, an error occurs when a job is submitted.	The requested type is the default type defined by the job ACL function. If the default value is not defined, it is treated as if the pertinent custom resource is not used.

See

To check the numerical value and type of custom resources defined by the job ACL function, see "2.2.2 Checking restriction information."

"As described in "1.8 Custom Resource," custom resources are defined as "custom resources per resource unit or resource group" or "custom resources per node." Therefore, the number of allocated custom resources varies depending on the type of submitted job, even if you have specified the same number of custom resources.

The following examples show specified custom resources per resource unit or resource group.

1. Normal job
   Only the specified numerical value of custom resources is allocated.

   ```
   $ pjsub -L "node=2,customrscname=3" ./job.sh
   ```

   customrscname: Custom resource name

In this example, the numerical value of allocated custom resources is three.

2. Step job
   The numerical value of allocated custom resources for a step job is greater than for a normal job when the step job has two or more sub jobs.

```
$ pjsub --step -L "node=2,customrscname=3" ./job1.sh,./job2.sh
```

   In this example, the numerical value of allocated custom resources is six (three of the custom resource multiplied by two sub jobs).

3. Bulk job
   Like with a step job, the numerical value of allocated custom resources for a bulk job is increased by the number of sub jobs.

```
$ pjsub --bulk --sparam 0-2 -L "node=2,customrscname=3" ./job1.sh
```

   In this example, the numerical value of allocated custom resources is nine (three of custom resources multiplied by three sub jobs).

When you have specified a custom resource per node, specify the method of allocating node resources. To do so, specify the requested quantity of custom resources per node or virtual node. Specify the node or vnode parameter, respectively. For details on how to specify the parameter, see "2.3.2.2 Specifying a node resource."

The following examples show custom resources per node.

1. Node allocated job (node specified)
   The numerical value of custom resources specified in node is multiplied by the number of nodes to allocate custom resources.

```
$ pjsub -L "node=2,customrscname=1" ./job.sh
```

   In this example, the numerical value of allocated custom resources is two (one per node multiplied by two nodes).

2. Virtual node allocated job (vnode specified)
   The numerical value of custom resources specified in vnode is multiplied by the number of virtual nodes to allocate custom resources.

```
$ pjsub -L "vnode=1,vnode-core=3,customrscname=3" ./job.sh
```

   Or

```
$ pjsub -L "vnode=1(core=3;customrscname=3)" ./job.sh
```

   In this example, the numerical value of allocated custom resources is three (three per virtual node multiplied by one virtual node).

2.3.2.5 Job operation when a job exceeds an upper limit on amount of resources

This section describes job operation in cases where the amount of resources used by a job exceeds a set upper limit during job execution.

Table 2.12 Operation of a job exceeding an upper limit on amount of resources

Resource	Operation
elapse	- When specified with the elapse=*limit* format The SIGXCPU signal is sent to all processes in the job. Then, after 10 seconds elapse, the SIGKILL signal is sent to any of these processes still existing in the job in order to forcibly terminate them. The purpose of this 10-second delay is to allow for the execution of postprocessing of every process. Therefore, implement processing as required by capturing the SIGXCPU signal from the job script and each process executed from the job script. - When specified with the elapse=*min_limit*-*max_limit* format The running job exceeds the minimum value *min_limit* for the elapsed time. Before reaching the maximum value *max_limit* for the elapsed time, the job may be forcibly terminated in order for execution of a subsequent job to begin. If so, SIGTERM is sent to all processes in the job at a certain time before the forced termination. The default time is 10 seconds in advance. The administrator can change this time.

Resource	Operation
	The SIGXCPU signal is sent to all processes in the job, like in the above case of elapse=*limit*, when the job reaches the maximum value *max_limit* for the elapsed time.
node-mem vnode-mem mem core-mem	The OS forcibly terminates a process that requested memory acquisition. The subsequent operation depends on the program and job script setup. The administrator may have configured the system to not only terminate the process but also forcibly terminate each job. If configured like this, the system runs OOM Killer when the OS memory is exhausted due to a job. Then, among the jobs that use this node, all the jobs that have the same user ID as the job causing the exhaustion are forcibly terminated. For details on the settings in your system, contact the administrator.
proc-*	Upper limit values such as proc-core are upper limit values established by the OS (system call setrlimit()) for individual processes in a job. Each of them corresponds to the following resources, which can be specified in the system call setrlimit(): proc-core: RLIMIT_CORE proc-cpu: RLIMIT_CPU proc-crproc: RLIMIT_NPROC proc-data: RLIMIT_DATA proc-lockm: RLIMIT_MEMLOCK proc-msgq: RLIMIT_MSGQUEUE proc-openfd: RLIMIT_NOFILE proc-psig: RLIMIT_SIGPENDING proc-filesz: RLIMIT_FSIZE proc-stack: RLIMIT_STACK proc-vmem: RLIMIT_AS The specified upper limit values are set as software and hardware limits on these resources. Operation when a process exceeds an upper limit value follows OS specifications. For details, see setrlimit(2) in the man page of Linux. Note that FX server operation conforms to Linux specifications.

Note

- The job resource limit values also apply to the prologue and epilogue processes set by the administrator (see "1.12 Prologue and Epilogue Function").
  The prologue and epilogue processes are executed as a part of a job. Therefore, even when a user-created job script or program does not exceed a job resource limit value, its prologue or epilogue process may exceed the limit value.
  For details on the amount of resources required for the prologue and epilogue processes, contact the administrator.
- When jobs are executed in the KVM mode (refer "2.3.8 Specifying a job execution environment") of the job execution environment, exceeding the memory resource set by the job management software is not detected. The behavior when the memory usage is excessive in a VM is depends on the setting of the VM image.

2.3.2.6 Specifying job statistical information output

You can output job statistical information as job execution results by using the -s or -S option of the pjsub command.

```
{ -s | -S } [ --spath output destination ]
```

The -S option outputs more detailed statistical information than the -s option. For details on their differences, see "A.2 Output of job statistical information" and man page pjstatsinfo(7).

Information

You can also use the long options --stats and --STATS as substitutes for the -s and -S options, respectively.

The job statistical information is output to the job statistical information file "*<job name>.<job ID>*.stats" file in the current directory at the job submission time. To specify an output destination, specify a file name in the --spath option.

The following expression can be used in the file name.

Table 2.13 Notation for specifying the file name of the job statistical information file

Notation	Meaning
%j	Job ID
%J	Sub job ID
%b	Bulk number
%s	Step number
%n	Job name

If you want to receive job statistical information by e-mail instead of in a file, use the -m option.

```
-m {s | S}
```

In this case, the arguments s and S have the same meanings as the -s and -S options, respectively.

2.3.2.7 Specifying automatically re-execution for a job

When submitting a job, you can specify whether to automatically re-execute any job aborted during execution, such as because of a system failure, by using the --restart or --norestart option of the pjsub command.

Table 2.14 Options specifying whether a job can be executed again

Option	Description
--restart	Automatically re-executes the job if it ended abnormally.
--norestart	Does not automatically re-execute the job, even if it ended abnormally.

Note

- If neither of these options is specified, operations in this case may change with the system settings. Ask the administrator about these operations on the system used by users.
- Interactive jobs cannot be automatically re-executed. If these options are specified for these jobs, they are ignored.

2.3.2.8 Specifying an execution start time

Normally, the jobs submitted by users are executed as soon as possible, with the execution order determined according to resource availability and priority. However, users can specify execution start times.

To specify an execution start time, use the --at option of the pjsub command in the following format.

Format	Description
--at *YYYYMMDD*[*hhmm*]	*YYYY* is the year, *MM* is the month, and *DD* is the day. *hh* is the hour, and *mm* is the minute. If *hhmm* is omitted, 00:00 is assumed specified. The specification in seconds is not available.

The following example shows the submission of a job with the execution start time specified.

```
$ pjsub --at 201908011511 job.sh   (*)Execution of job.sh starts at 15:11 on August 1, 2019.
```

Note

- A job with the execution start time specified is never executed before the specified time even when there are free resources. Furthermore, it may be executed later than the specified time, depending on the availability of resources.
- An interactive job cannot have a specified execution start time. If one is specified, it is ignored.

2.3.2.9 Specifying a job priority

Users can set execution priority for only the jobs that they submitted. Job priority is specified in the -p option of the pjsub command. The specified priority is an integer in a range of 0 to 255.

```
$ pjsub -p priority job.sh
```

The lowest priority is 0, and the highest is 255.

See

Users can check the priorities of their own jobs by using the -v option of the pjstat command. For details, see "2.4.1 Displaying a job list."

Information

- Any jobs that have the same priority are executed in order of submission.
- If the -p option is not specified, the priority is determined according to the administrator's settings.

2.3.2.10 Specifying the standard output and standard error output files of a batch job

The standard output and standard error output of a batch job are output to files.
The file names are shown below. For details, see "2.6.1 Referencing job execution results."

- Standard output: *<job name>.<job ID>*.out
- Standard error output: *<job name>.<job ID>*.err

The -o option and -e option of the pjsub command are used by users who want to specify the file names. To direct the standard error output to the standard output, specify the -j option.

Table 2.15 Specifying the standard output and standard error output files of a batch job

Format	Description
-o *filename*	Outputs the standard output of the job to *filename* file.
-e *filename*	Outputs the standard error output of the job to *filename* file.
-j	Directs the standard error output of the job to the standard output of the job.

Information

You can also use the long options --out and --err as substitutes for the -o and -e options, respectively.

You can use the following notation to specify the output file names.

Notation	Meaning
%j	Job ID
%J	Sub job ID

Notation	Meaning
%b	Bulk number
%s	Step number
%n	Job name

The following example shows the specification of output file names using the above notation.

```
$ pjsub -o '%j[%b].stdout' -e '%j[%b].stderr'
```

In this example, a sub job with bulk number 10 in a bulk job with job ID 100 has the following file names. The standard output file is 100[10].stdout, and the standard error output file is 100[10].stderr.

- If individual sub jobs in a bulk job or step job have the same file specified for their output destinations, the file will contain a mixture of output from each sub job. Consequently, the output from each sub job may not be readable in the output results in the file.
- The -o and -e options of the pjsub command cannot be specified for an interactive job.
- When you execute the mpiexec command in a job that runs on the FX server, its standard output and standard error output are written to the file defined by the item 'mpiexec-*' of the job ACL function (see "2.2.2 Checking restriction information") unless you specify otherwise with the mpiexec command option. As a result, job stdout and stderr output files may be created in addition to the files specified by the -o and -e options of the pjsub command.
  For more information about the standard output and standard error output of the mpiexec command, see "2.3.6.9 Standard output/standard error output of the mpiexec command [FX]" and the Development Studio manual"MPI User's Guide."

2.3.2.11 Allocating resources in consideration of I/O nodes [FX]

Allocating compute nodes in consideration of the position of the I/O node (storage I/O node or global I/O node) that processes I/O of jobs on the Tofu coordinate is expected to suppress the fluctuation of the I/O performance of the job and improve the I/O performance.

There are following types of allocating nodes with considering I/O nodes:

- Allocate so that the distance between the I/O node and the compute node is the same each time.

  This allocation method has the effect of preventing I/O performance from fluctuating each time a job is run.

- Allocate I/O nodes to be occupied by a job

  This allocation method has the effect that one job occupies I/O nodes and is not affected by the I/O of other jobs.

Each is described below.

[Allocate so that the distance between the I/O node and the compute node is the same each time]

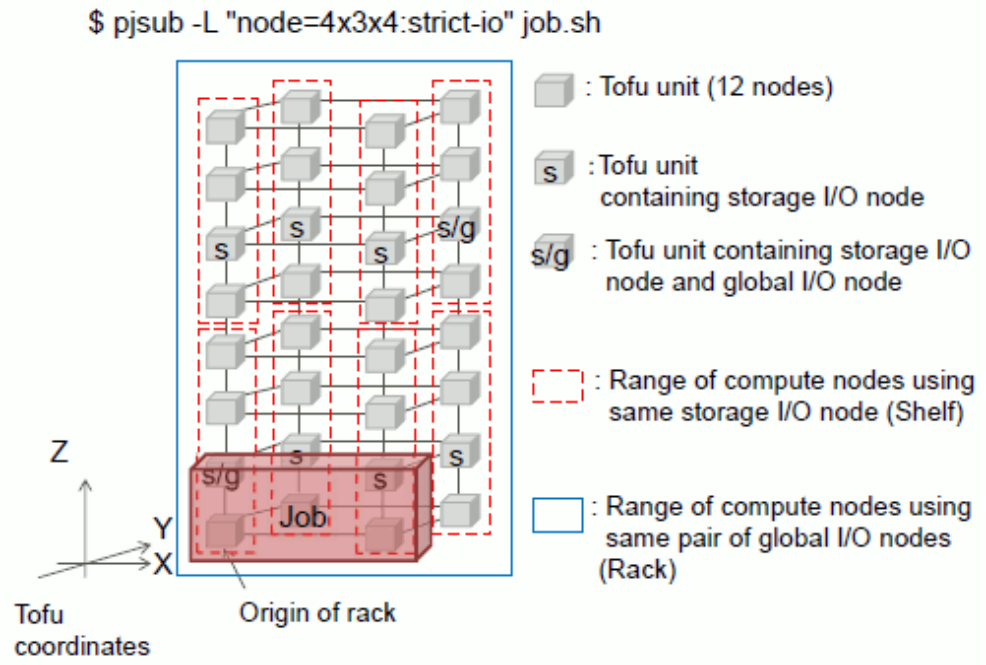
The I/O performance of a job is affected by the distance (difference of coordinate) between the Tofu coordinates of the nodes executing the job and the I/O node processing input/output.
With the strict-io parameter specified, nodes are placed on Tofu coordinates without rotating the specified shape. The start point of the node allocation is always the origin coordinate of the FX server main unit rack (minimum coordinates in the rack). As a result, the location of the I/O node for each compute node is the same every time, and I/O performance of a job does not fluctuate each time the job is run.

```
$ pjsub -L "node=N1xN2xN3:torus:strict-io" job.sh
```

You can specify the strict-io parameter only when allocating nodes in torus mode and in a three-dimensional shape.
Note that the following explanation assumes that the torus mode is set as the default value by the job ACL function and omits it from the command-line examples.

The following figure shows an example of allocating a node of shape 4x3x4 with the parameter strict-io. The node shape 4x3x4, expressed in 6-dimensional Tofu coordinates (X, Y, Z, A, B, C), is (X axis:2, A axis:2) x (Y axis:1, B axis:3) x (Z axis:2, C axis:2). This is equivalent to a box in Tofu units, with the sides of the X, Y, and Z axes having lengths of 2, 1, and 2, respectively, in Tofu coordinates. It is allocated starting at the rack origin.

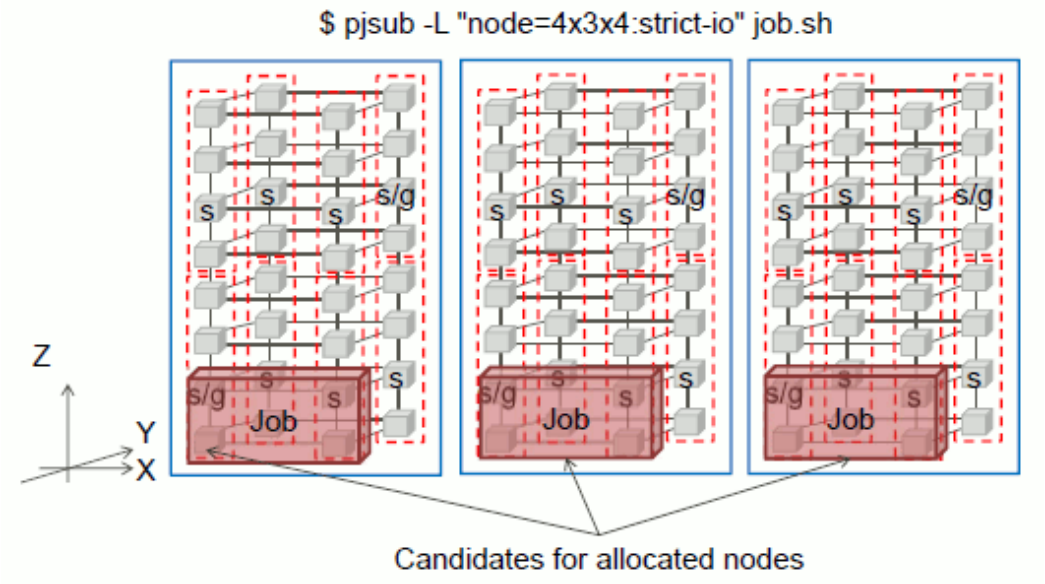
Figure 2.4 Node allocation with the parameter strict-io



Information

For example, in a three-rack system, if you try to allocate 4x3x4 nodes with parameter strict-io to a job, there are three possible locations to place as shown in the following figure. The job operation software determines which one to select. Even if the rack allocated each time is different, the position of the I/O node for each compute node is the same each time because origin coordinate of the rack is the starting point.

Figure 2.5 The example of multiple candidate allocation nodes



[Allocate I/O nodes to be occupied by a job]

To maximize I/O performance, I/O nodes should be occupied by a job being submitted, not just the I/O node location. To do this, allocate the nodes in the range that the I/O node is responsible for I/O as a unit, as described below.

- Using storage I/O nodes exclusively

  Occuping storage I/O nodes by a job maximizes its I/O performance of the first-layer storage (LLIO) in FX server. This is called I/O-exclusive mode, and a job in I/O-exclusive mode is called an I/O-exclusive job. In I/O-exclusive mode, allocating nodes to a job on a shelf by shelf basis (48 nodes) in the FX server allows only that job to use storage I/O node exclusively, independent of other job inputs and outputs.
  Specify the parameter io-exclusive when submitting the job for I/O-exclusive mode.

```
$ pjsub -L "node=shape:io-exclusive" job.sh
```

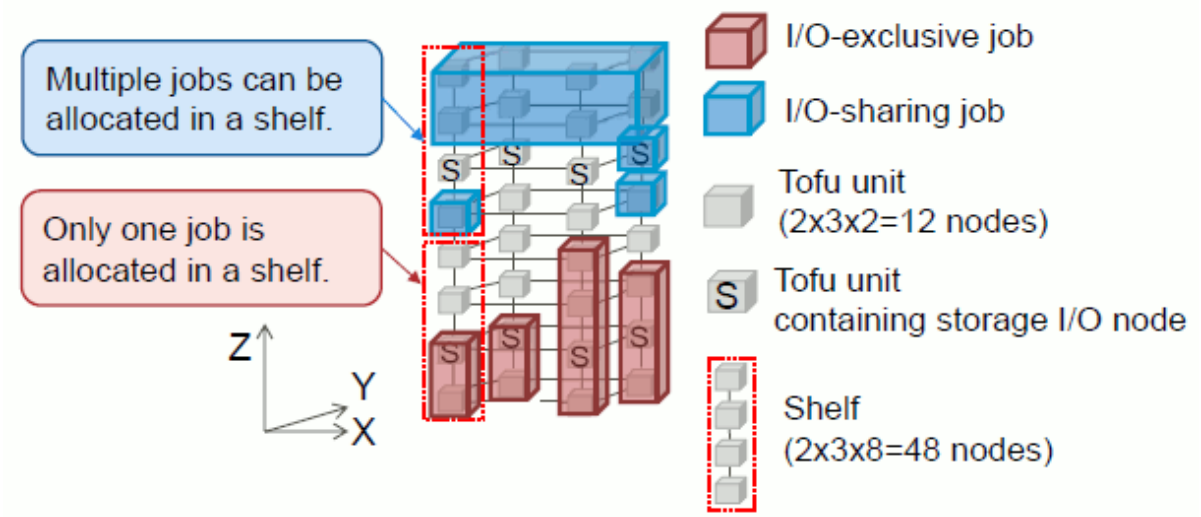
  I/O-exclusive mode can be specified with torus mode, but not mesh or noncont mode.
  A mode that does not occupy storage I/O nodes is called I/O-sharing mode, and a job in I/O-sharing mode is called an I/O-sharing job.
  Specify the parameter no-io-exclusive when submitting the job for I/O-sharing mode.

```
$ pjsub -L "node=shape:no-io-exclusive" job.sh
```

Information

No other job is allocated in a shelf that has a I/O-exclusive job allocated. Also, shelf used by other jobs is not allocated to the I/O-exclusive job.
Free nodes in a shelf used by an I/O-sharing job may be allocated another I/O-sharing job.

Figure 2.6 I/O-exclusive mode and I/O-sharing mode



  In the figure above, there are multiple positions of nodes allocated to the I/O-exclusive job, but if you also specify the parameter strict-io in addition to the parameter io-exclusive, only the node starting at rack origin is always allocated.

- Using global I/O nodes exclusively

  To occupy global I/O nodes, you must allocate 4x6x16 nodes (a rack) as an unit. This unit is the range of nodes for which the two global I/O nodes in the pair process input and output. In this case, storage I/O nodes in the rack are also occupied. And, specify the parameter strict-io to allocate starting at the rack origin.

```
$ pjsub -L "node=4x6x16:strict-io" job.sh
```

If you do not specify the strict-io, io-exclusive, and no-io-exclusive parameters, the default values set in the job ACL function are applied. In addition, the administrator might have set these parameters to be unavailable through the job ACL function.

2.3.2.12 Specifying the operation when a Tofu interconnect link is down [FX]

If a Tofu interconnect link goes down due to a hardware failure or other cause during job execution on an FX server, communication between job processes is affected. However, this may be avoided by dynamically changing the communication path.
With the --net-route option of the pjsub command, you can specify whether to change the communication path when a Tofu interconnect link goes down.

Table 2.16 --net-route option of the pjsub command

Option	Description
--net-route dynamic	Changes the communication path when a Tofu interconnect link goes down. Job execution continues.
--net-route static	Does not change the communication path when a Tofu interconnect link goes down. The job ends abnormally.

You can specify the --net-route option for node-exclusive jobs on an FX server and MPI jobs (only for MPI processing using Development Studio). If specified for a job on a PRIMERGY server, the option is ignored.

If the option is not specified, the job operates according to the value of the define net-route item of the job ACL function. You can specify the --net-route option only when the execute pjsub-net-route item of the job ACL function is set to enable. For details on how to check the job ACL function settings, see "2.2.2 Checking restriction information."

If the --net-route dynamic option is specified, the job operates as follows when a Tofu interconnect link goes down.

- The communication path is changed by the Tofu library, and communication is retried (retransmission).
  Then, a message indicating the retransmission is output to the standard error output of the job.
  If this message does not need to be output, set the environment variable UTOFU_LINKDOWN_INFO to 0 in the job script.
- If the Tofu library judges that job execution cannot continue even when the communication path has changed, an error message is output to the standard error output file of the job. Then, the job is terminated. The end code (PJM code) of the job at that time is 22.

For details on output messages, see "Tofu Interconnect" in "Messages in job outputs" of the "Job Operation Software Command Reference" manual.

Note the following about specifying the --net-route dynamic option.

- Job execution performance may deteriorate. Be careful about using it when job execution performance is critical.
- The barrier communication function provided as a Tofu interconnect hardware function cannot be applied. For details, see the "MPI User's Guide," which is a Development Studio manual.
- Depending on the location of the link that went down, there may be no alternative communication path, in which case job execution cannot continue.

2.3.3 Specifying LLIO-related parameters [FX]

When submitting a job, you can specify LLIO-related parameters in the --llio option of the pjsub command.

```
$ pjsub --llio param=arg,...
```

LLIO-related parameters are the parameters that determine the size of areas in first-layer storage and the behavior of the layered storage system. If the parameters are not specified, the size and behavior depend on the job ACL function settings.

Figure 2.7 Layered storage system

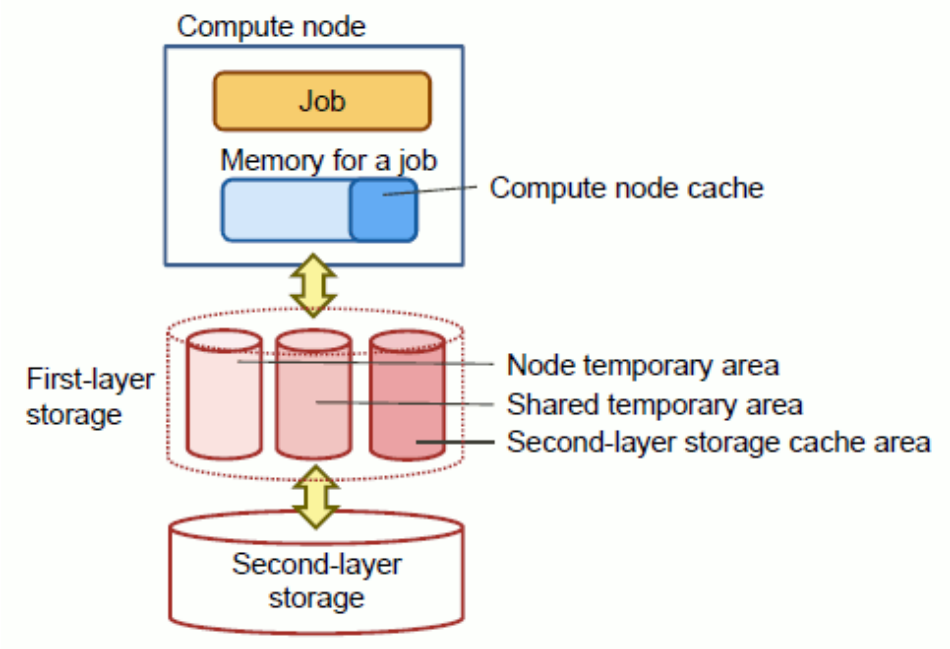


Table 2.17 Parameters related to areas in first-layer storage

Parameter	Description
localtmp-size=*size*	Size of the node temporary area on first-layer storage For details, see "2.3.3.1 Size of first-layer storage."
sharedtmp-size=*size*	Size of the shared temporary area on first-layer storage (*) Multiply *size* by the number of compute nodes, and the resulting value is the size of the shared temporary area available to one job. For details, see "2.3.3.1 Size of first-layer storage."
sio-read-cache={on|off}	Operation of whether or not to cache a file in first-layer storage when reading the file from second-layer storage to a compute node on: Cache. off: Do not cache. For details, see "2.3.3.2 Caching a file read from second-layer storage."

The size of the second-layer storage cache area depends on the sizes of the node temporary area and shared temporary area. For details, see "2.3.3.1 Size of first-layer storage."

Table 2.18 Striping-related parameters for first-layer storage

Parameter	Description
stripe-count=*count*	Number of stripes per file when files are distributed across first-layer storage For details, see "2.3.3.3 Striping."
stripe-size=*size*	Stripe size when files are distributed across first-layer storage For details, see "2.3.3.3 Striping."

Table 2.19 I/O operation-related parameters for first-layer storage and second-layer storage

Parameter	Description
async-close={on|off}	Operation of whether or not to asynchronously close files on first-layer storage and second-layer storage

Parameter	Description
	on: Asynchronous close off: Synchronous close The setting of asynchronous close ("on") does not guarantee the completion of writing when a file is closed. However, the completion of writing is guaranteed when a job ends. The setting of synchronous close ("off") guarantees writing completion. For details, see "2.3.3.4 Asynchronous close."
auto-readahead={on|off}	Operation of whether or not to automatically read ahead when a job is going to try reading consecutive areas again on first-layer storage or second-layer storage multiple times on: Read ahead. off: Do not read ahead. For details, see "2.3.3.5 Automatic readahead of files."

Table 2.20 Parameters related to compute node cache

Parameter	Description
cn-cache-size=*size*	Compute node cache size on first-layer storage For details, see "2.3.3.6 Compute node cache size."
cn-cached-write-size=*size*	Threshold value of whether or not to cache data when writing the data to first-layer storage When writing data to first-layer storage, if the write size is equal to or less than the specified value, the data is not immediately written to the storage. Instead, it is temporarily cached in compute node cache. This parameter improves performance by outputting smaller files collectively, reducing the number of file transfers to first-layer storage. For details, see "2.3.3.7 Cache threshold value when writing to first-layer storage."
cn-read-cache={on|off}	Operation of whether or not to cache a file in compute node cache when reading the file from first-layer storage or second-layer storage on: Cache. off: Do not cache. For details, see "2.3.3.8 Caching a file read from a compute node."

Table 2.21 Parameters related to I/O statistical information for the layered storage system

Parameter	Description
perf[,perf-path=*path*]	LLIO performance information is output to a file. The output destination is under the current directory at the job submission time. The file name is defined by the job ACL function. You can also specify the output destination file in the perf-path parameter. For details, see "2.3.3.9 Collecting LLIO performance information."
uncompleted-fileinfo-path=*path*	Output destination for information on files that have not completed writing from cache to second-layer storage at job end For details, see "2.3.3.10 Information on uncompleted files2.3.3.10 Information on uncompleted files."

The following sections describe each parameter.

2.3.3.1 Size of first-layer storage

First-layer storage contains the node temporary area and shared temporary area, which are available as high-speed temporary storage to running jobs.
When submitting a job, you can specify the size of the node temporary area or shared temporary area used by the job. In the localtmp-size parameter, specify the size of the node temporary area per node. In the sharedtmp-size parameter, specify the shared temporary area size

divided by the number of allocated nodes. These parameters are in the --llio option of the pjsub command. If no size is specified, the default value that is set by the job ACL function is applied.

Note

The value specified by the sharedtmp-size parameter is not the size of the entire shared temporary area available to jobs. The value specified in the sharedtmp-size parameter multiplied by the number of nodes allocated to a job is the size of the entire shared temporary area.

```
$ pjsub --llio localtmp-size=size,sharedtmp-size=size job.sh
```

Check the localtmp-size and sharedtmp-size items displayed by the pjacl command to see the upper limit, lower limit, and default value for the values that can be specified as the size of each area.

```
$ pjacl
...
pjsub option parameters
   ...
    (--llio)                              lower           upper            default
        (sharedtmp-size=)                 0Mi             2147483647Mi     0Mi
        (localtmp-size=)                  0Mi             2147483647Mi     0Mi
   ...
```

The first-layer storage area is secured from the SSD (Solid State Drive) of a storage I/O node taking care of input/output in the FX server. The SSD size allocated to one compute node is a value equally divided by the number of compute nodes whose input/output is taken care of by one storage I/O node.

```
First-layer storage size allocated to 1 compute node
= <SSD size of storage I/O node>
  / <number of compute nodes for which 1 storage I/O node takes care of input/output>
```

For example, suppose that the SSD size is 800GiB and 1 storage I/O node takes care of input/output for 16 compute nodes. Then, the first-layer storage size allocated to one compute node is 50GiB. Both the SSD size of the storage I/O node and the number of compute nodes for which one storage I/O node takes care of input/output vary depending on the system. For details, contact the administrator.

Among the SSD areas allocated to one compute node, areas other than the node temporary area and shared temporary area compose the second-layer storage cache per node. However, since at least 128 MiB per compute node is required for the cache of the second-layer storage, in the above example, it is possible to specify the total size of node temporary area and shared temporary area up to (50GiB-128MiB). The cache of the second-layer storage that one job can use is the size of this value multiplied by the number of nodes.

Note

- If the size of the node temporary area (localtmp-size) is 0, you cannot create files on the node temporary area.
  If the size of the shared temporary area (sharedtmp-size) is 0, you can create files but cannot store files' data on the shared temporary area.
- The node temporary area and the shared temporary area are secured on high-speed storage (SSD) of the same storage I/O node. For this reason, if their total size exceeds the size of the storage, an error is returned at the job submission time regardless of whether their individual size is equal to or less than the upper limit. The error message displays the size that is the upper limit for the total value.
- If the areas have no free space, the write system call returns an error in ENOSPC.
  Shared temporary area and node temporary area are consumed not only by data of files but also by meta data of files. Therefore, file access might fail with ENOSPC even if the total amount of data of files included in the filesystem does not exceed the specified amount of these areas. The amount of meta data of files can be estimated as follows.

  ```
  300bytes * <The number of files>
  ```

- Even though you do not run out of all the available space of the shared temporary area specified when the job was submitted, you may encounter errors due to insufficient free space.
  This may be because a job uses more than one storage I/O node(*) and runs out of available space on one storage I/O node. If this occurs, increase the amount of shared temporary space and run the job.

(*) You can verify that a job is utilizing multiple storage I/O nodes by checking the LLIO performance information (see "2.3.3.9 Collecting LLIO performance information") for multiple storage I/O nodes.

- The sizes of the node temporary area and shared temporary area have a minimum unit of 1MiB. Specify their sizes in powers of 2 (Ki, Mi, Gi, etc.), in the same way as amounts of memory.
  Although the sizes of disk units are generally represented as a power of 10 (K, M, G, etc.), the above examples represent SSD sizes as a power of 2 in order to simplify the description. Take care with the difference in units when calculating a size.

2.3.3.2 Caching a file read from second-layer storage

If a job reads the same file multiple times, the file is cached on first-layer storage when the job reads the file on second-layer storage. This can be expected to improve performance.

You can specify whether to cache on first-layer storage in the sio-read-cache parameter in the --llio option of the pjsub command.

```
$ pjsub --llio sio-read-cache={on|off} job.sh
```

If sio-read-cache=on is specified, a file read from second-layer storage is cached on first-layer storage. If sio-read-cache=off is specified, the read file is not cached.

If no parameter is specified, the behavior depends on the job ACL function settings. Check the default llio-sio-read-cache item displayed by the pjacl command.

```
$ pjacl
...
defines
   ...
    default llio-sio-read-cache off
```

2.3.3.3 Striping

Using striping in first-layered storage has the following advantages:

- You can create a file whose size exceeds the physical capacity of a single storage device (SSD) connected to the storage I/O node.
- Distributing and storing single files across multiple first-layer storage improves the file access bandwidth.

To use striping, specify the number of stripes in the stripe-count parameter and the stripe size in the stripe-size parameter. The parameters are in the --llio option of the pjsub command.

```
$ pjsub --llio stripe-count=count,stripe-size=size job.sh
```

You can specify values within the respective ranges set by the job ACL function. Check the stripe-count and stripe-count items displayed by the pjacl command.

```
$ pjacl
...
pjsub option parameters
   ...
    (--llio)                              lower           upper           default
   ...
        (stripe-count=)                   1               2147483647      2147483647
        (stripe-size=)                    64Ki            4194240Ki       2048Ki
```

The stripe size has a minimum unit of 64KiB. Specify it in powers of 2 (Ki, Mi, Gi, etc.), in the same way as amounts of memory.

If the stripe size of first-layer storage is not a multiple of the stripe size of second-layer storage, the performance of second-layer storage may not be gained. You can obtain the stripe size and stripe count of second-layer storage by using the getstripe sub command of the lfs command. You can change them by using the setstripe sub command of the lfs command. However, both of these operations must be

performed within a job. Confirm the specification of the second-layer storage (FEFS) about available upper and lower limit of the stripe size and the stripe counts.

The following example shows a job script to change the stripe size of second-layer storage. This example uses the getstripe sub command of the lfs command at the end to obtain stripe-related information about second-layer storage.

```
#!/bin/bash
#
#PJM --llio sharedtmp-size=5Mi,localtmp-size=10Gi
#PJM --llio stripe-size=1Mi
lfs setstripe -S 1m /gfs1/user1/data    <- Set stripe size of second-layer storage
mpiexec /gfs1/user1/a.out               <- Execute a.out program
lfs getstripe /gfs1/user1/data      <- Obtain stripe information about second-layer storage
```

After the getstripe sub command of the lfs command obtains stripe information about second-layer storage, the information is output to the standard output of the job as shown below.

```
/gfs1/user1/data
stripe_count:   1 stripe_size:    1048576 stripe_offset:  0
/gfs1/user1/data/file1
lmm_stripe_count:   1
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  0
        obdidx           objid           objid           group
             0           19538         0x4c52               0
```

For details on the getstripe and setstripe sub commands of the lfs command, see the "Reference" appendix in "LLIO User's Guide."

2.3.3.4 Asynchronous close

With the asynchronous close of the file close process, a program can proceed to the next process without waiting for writing to complete every time it closes a file.

Specify whether to asynchronously close files with the async-close parameter in the --llio option of the pjsub command.

If async-close=on is specified, the close process is asynchronous close. If async-close=off is specified, the close process is synchronous close.

```
$ pjsub --llio async-close={on|off} job.sh
```

If no parameter is specified, the behavior depends on the job ACL function settings. Check the default llio-async-close item displayed by the pjacl command.

```
$ pjacl
...
defines
   ...
    default llio-async-close    on
```

Even when asynchronous close (async-close=on) is specified, jobs end after waiting for writing to complete.

The wait time for this writing is included in the job elapsed time.

2.3.3.5 Automatic readahead of files

By reading ahead files to compute node cache when reading the files on the layered storage system, programs are expected to reach high speeds if they read consecutive areas.

In the auto-readahead parameter in the --llio option of the pfsub command, specify whether programs automatically read ahead consecutive areas to compute node cache.

Use auto-readahead=on to automatically read ahead. Use auto-readahead=off to not automatically read ahead.

```
$ pjsub --llio auto-readahead={on|off} job.sh
```

Note

Automatic readahead is disabled if the cn-read-cache=off parameter is specified.

If no parameter is specified, the behavior depends on the job ACL function settings. Check the default llio-auto-readahead item displayed by the pjacl command.

```
$ pjacl
...
defines
   ...
    default llio-auto-readahead on
```

2.3.3.6 Compute node cache size

High-speed input/output can be expected from the use of free memory as compute node cache when allocated memory has more free space.

Specify the maximum size of the compute node cache in the cn-cache-size parameter of the pjsub command.

```
$ pjsub --llio cn-cache-size=size job.sh
```

Check the cn-cache-size item displayed by the pjacl command to see the upper limit, lower limit, and default value for the values that can be specified.

```
$ pjacl
...
pjsub option parameters
   ...
    (--llio)                              lower           upper           default
   ...
        (cn-cache-size=)                  4Mi             2147483647Mi    128Mi
```

Note

- When the specified size is the upper limit, compute node cache is secured from the memory allocated to a job. Consequently, if the specified compute node cache is larger than necessary, a job may fail to acquire the memory required by processes in the job. Be careful.
- If the cn-read-cache parameter discussed below is set to on, read files are cached in the compute node.
  The data smaller than the cn-cached-write-size parameter discussed below are cached in the compute node cache.
- Specify sizes in powers of 2 (Ki, Mi, Gi, etc.), in the same way as amounts of memory.

2.3.3.7 Cache threshold value when writing to first-layer storage

When a job write to a file, the LLIO caches smaller data volumes temporarily in compute node cache and then transfers them collectively to first-layer storage later. This can be expected to affect performance.

You can specify the threshold value of this size in the cn-cached-write-size parameter in the --llio option of the pjsub command. For the writing of data volumes smaller than the threshold value, they are temporarily cached in compute node cache.

```
$ pjsub --llio cn-cached-write-size=size job.sh
```

Note

- If the specified threshold value is larger than the compute node cache size (--llio cn-cache-size), job submission returns an error.
- The threshold value has a minimum unit of 4KiB. Specify sizes in powers of 2 (Ki, Mi, Gi, etc.), in the same way as amounts of memory.

Check the cn-cached-write-size item displayed by the pjacl command to see the upper limit, lower limit, and default value for threshold values that can be specified.

```
$ pjacl
...
pjsub option parameters
   ...
    (--llio)                                lower           upper           default
   ...
        (cn-cached-write-size=)             0Mi             2147483647Mi    4Mi
```

2.3.3.8 Caching a file read from a compute node

A file is cached in a compute node cache during reading of the file from first-layer storage or second-layer storage to the compute node. This can be expected to improve performance if the file is read repeatedly.

In the cn-read-cache parameter in the --llio option of the pfsub command, specify whether to cache a file in a compute node cache when reading the file from the layered storage system.

Specify cn-read-cache=on to cache the read file in a compute node. Specify cn-read-cache=off to not cache it.

```
$ pjsub --llio cn-read-cache={on|off} job.sh
```

If no parameter is specified, the behavior depends on the job ACL function settings. Check the default llio-cn-read-cache item displayed by the pjacl command.

```
$ pjacl
...
defines
   ...
    default llio-cn-read-cache    on
```

2.3.3.9 Collecting LLIO performance information

You can collect LLIO performance information to subsequently analyze the status of access to first-layer storage and tune jobs.

Specify the perf parameter in the --llio option of the pjsub command to output the LLIO performance information to a file under the current directory at the job submission time.

```
$ pjsub --llio perf job.sh
```

The file name is defined by the job ACL function. If the perf parameter is not specified, whether or not the LLIO performance information is output depends on the job ACL function definitions. Check the default llio-perf-path and default llio-perf items displayed by the pjacl command to see these job ACL function definitions.

```
$ pjacl
...
defines
   ...
    default llio-perf           off
    default llio-perf-path      %n.%J.llio_perf
```

The file name format uses the following metacharacters.

Table 2.22 Metacharacters for file names

Metacharacter	Description
%j	Deployed for job ID
%J	Deployed for sub job jobID number
%b	Deployed for bulk number
%s	Deployed for step number
%n	Deployed for job name (up to 63 characters) For jobs submitted from the standard input, "STDIN" is output. If a job name begins with a single-byte number, the letter "J" is added to the beginning of "job name."
%o	Deployed for standard output file path. Deployed for "./%n.%J.out" for an interactive job.
%e	Deployed for standard error output file path. Deployed for "./%n.%J.err" for an interactive job.

You can also specify the path of the output file in the perf-path parameter.

```
$ pjsub --llio perf,perf-path=path job.sh
```

You can use metacharacters shown in "Table 2.22 Metacharacters for file names" for the file name specified in the perf-path parameter.

The LLIO performance information output to the file is as follows:

```
I/O     2ndLayerCache   NodeTotal                       Count         Amount(Byte)         Time(us)
                Write(Cached I/O)       Sum             1             15                   236
                                        [1,4Ki)         1             15                   236
                                        [4Ki,1Mi)       0             0                    0
                                        [1Mi,4Mi)       0             0                    0
                                        [4Mi+           0             0                    0
                Read(Cached I/O)        Sum             12            1827                 12987
                                        [1,4Ki)         12            1827                 12987
                                        ...
I/O     2ndLayerCache   ComputeNode<->CommBuf           Count         Amount(Byte)         Time(us)
                Write(Cached I/O)       Sum             -             -                    68
...
Resource        LocalTmp        CacheUsage              Amount(Byte)
                                        MaximalUsedSpace            0

SIO Information
Node ID : 0x01010002
I/O     2ndLayerCache   NodeTotal                       Count         Amount(Byte)         Time(us)
                Write(Cached I/O)       Sum             1             15                   236
                                        [1,4Ki)         1             15                   236
                                        [4Ki,1Mi)       0             0                    0
                                        [1Mi,4Mi)       0             0                    0
                                        [4Mi+           0             0                    0
                Read(Cached I/O)        Sum             12            1827                 12987
                                        [1,4Ki)         12            1827                 12987
                                        ...
I/O     2ndLayerCache   ComputeNode<->CommBuf           Count         Amount(Byte)         Time(us)
                Write(Cached I/O)       Sum             -             -                    68
                ...
...
```

See

For more information on the LLIO performance information, see "Output Items of LLIO Performance Information" in the appendix "Statistical Information Output Items" of the manual "LLIO User's Guide."

Information

- Statistical information about first-layer storage is also included in the job statistical information output by the -s/-S option of the pjsub command and the -s/-S option of the pjstat command. For details, see the pjstatsinfo(7) man page.
- The LLIO provides the library function getlliostat(). By creating an application using this getlliostat() function, you can obtain statistical information related to use of first-layer storage during a job. For details on how to use the getlliostat() function and the statistical information that can be obtained, see "Collection Methods and Output Items of Compute Node Statistical Information" in the appendix "Statistical Information Output Items" of the manual "LLIO User's Guide."

Note

LLIO performance information is collected while the job is in the RUNOUT state.

As the number of nodes executed the job increases, the time of collecting LLIO performance information also increases. Therefore the time of RUNOUT state increases as well.

Collecting LLIO performance information is finished in a few minutes, if the number of nodes for the job is 10,000. But it takes 30 miniutes for 20,000 nodes, and about 2 hours for 50,000 nodes.

2.3.3.10 Information on uncompleted files

Writing from first-layer storage to second-layer storage may not be able to complete by the time that the job ends. For example, write processing is interrupted upon reaching the elapsed time limit for the job. In this case, an analysis of information on uncompleted files can give a guide to adjusting the elapsed time limit value for a job.

You can output information on uncompleted files by using the uncompleted-fileinfo-path parameter in the --llio option of the pjsub command.

```
$ pjsub --llio uncompleted-fileinfo-path=path job.sh
```

In *path*, specify the output destination of information on uncompleted files. You can use the following metacharacters for the output destination.

Table 2.23 Specifiable metacharacters

Metacharacter	Description
%j	Deployed for job ID
%J	Deployed for sub job ID
%b	Deployed for bulk number
%s	Deployed for step number
%n	Deployed for job name (up to 63 characters) For jobs submitted from the standard input, "STDIN" is output. If a job name begins with a single-byte number, the letter "J" is added to the beginning of "job name."
%o	Deployed for standard output file path. Deployed for "./%n.%J.out" for an interactive job.
%e	Deployed for standard error output file path. Deployed for "./%n.%J.err" for an interactive job.

If no parameter is specified, the behavior depends on the job ACL function settings. Check the default llio-uncompleted-fileinfo-path item displayed by the pjacl command.

```
$ pjacl
...
defines
   ...
    default llio-uncompleted-fileinfo-path  %e
```

The following information is output for an uncompleted file.

```
<file transfer error information>  # Title line
<summary>                          # Title line of summary information
total num: 3                       # Number of transfer-failed files
total size: 128000000              # Transfer failure size [B]
error: E1,E3,E8                    # Error information
<detail>                           # Title line of detailed information
E1 /gfs1/userA/outfile1            # Error number file path
E1 /gfs1/userA/dir/outfile2
E3,E8 /gfs1/userA/dir/outfile3
```

Table 2.24 The output information

Type	Output item	Description of output
Summary <summary>	total num	Number of uncompleted files If this information cannot be obtained about any file, "+" is added after the numerical value. If the number of uncompleted files exceeds 1000, "1000*" or "1000*+" is output.
	total size	Total size of uncompleted files If this information cannot be obtained about any file, "+" is added after the numerical value.
	error	Error information (Reason why writing did not complete) E1: Second-layer storage QUOTA exceeded E2: Writing interruption E3: Writing timeout E4: Writing error E5: Shortage of free space in device E8: Other error
Detailed information <detail>	1st field	Error information (Reason why writing did not complete) E1: Second-layer storage QUOTA exceeded E2: Writing interruption E3: Writing timeout E4: Writing error E5: Shortage of free space in device E8: Other error
	2nd field	File path

If the number of uncompleted files exceeds 1000, only the following message appears in the detailed information.

```
The detailed information is not output because the number of uncompleted files exceeds the upper limit
(1000).
```

2.3.4 How to submit each type of job

Jobs other than normal jobs require specific specifications according to the type of job. This section describes differences between specification methods by type of job.

2.3.4.1 How to submit a bulk job

Specify the --bulk option of the pjsub command to declare the job as a bulk job.

Also, indicate the number of sub jobs you want to submit as a bulk job by using the --sparam option with start and end values for the bulk number.

```
$ pjsub --bulk --sparam "StartingBulkNumber-EndingBulkNumber" [jobscript]
```

You can specify a value from 0 to 999999 for a bulk number. The ending bulk number must be greater than the starting bulk number. The bulk number is incremented by one.

The following example shows the submission of a bulk job.

```
Program executed with each of ten data files, in-0.dat to in-9.dat, as input
$ cat bulkjob.sh
#!/bin/sh
...
INFILE=in-${PJM_BULKNUM}.dat                    (1)
OUTFILE=out-${PJM_BULKNUM}.dat                  (2)
./program ${INFILE} ${OUTFILE}                  (3)
...

$ pjsub --bulk --sparam "0-9" bulkjob.sh        (4)
```

(1) Determines the input data file name from the bulk number.
(2) Determines the output data file name from the bulk number.
(3) Specifies the input/output data files in the arguments of the program.
(4) Submits the sub job bulkjob.sh with the bulk number from 0 to 9.

The above example shows only the part related to the bulk job.

2.3.4.2 How to submit a step job

Specify the --step option of the pjsub command to declare the job as a step job.
For the step job, specify the job ID or the job name. It indicates which the sub job is associated with the step job.

[Method of specifying the job ID]

Submit a sub job with the job ID of the step job. In this case, the options of the first sub job differ from the options of the second and subsequent sub jobs.

1. Submission of the first sub job

```
$ pjsub --step stepjob1.sh
[INFO] PJM 0000 pjsub Job 100_0 submitted.
```

2. Submission of the second and subsequent sub jobs

   Specify a job ID with the --sparam "jid=" option to indicate that the job is associated with the first sub job.

```
$ pjsub --step --sparam "jid=100" stepjob2.sh
[INFO] PJM 0000 pjsub Job 100_1 submitted.
```

   If the first sub job has already ended, the sub job submission will fail because the corresponding job does not exist.

[Method of specifying the job name]

Make the sub job name same and submit sub job with the job name specified. In this method, the first job and the jobs after that can be submitted by same way basically.

1. Submission of the first sub job

   Set the job name with the --sparam "jnam=" option when submitting a sub job.

```
$ pjsub --step --sparam "jnam=mystepjob" stepjob1.sh
[INFO] PJM 0000 pjsub Job 200_0 submitted.
```

2. Submission of the second and subsequent sub jobs

   The --sparam "jnam=" option is for setting the job name, and means that the sub job is associated with an existing step job of same name.

```
$ pjsub --step --sparam "jnam=mystepjob" stepjob2.sh
[INFO] PJM 0000 pjsub Job 200_1 submitted.
```

   If the step job which has the corresponding job name does not exist, new step job will be created.

Note

Note the following when you use the --sparam "jnam=" option.

- If --sparam "jnam=" option is specified, the job name specified with the option is set to the sub job regardless of the -N or --name option.
- If the --sparam "jnam=" option is specified to the pjsub command, --sparam "jnam=" option in the job script is ignored.
- If you submit a sub job with the job name for the step job which has been submitted without --sparam "jnam=" option, specify the job name of the first sub job (step number 0).
- If you specify the job name with -N or --name option, not with the --sparam "jnam=" option, new step job will be created. Therefore, two or more step jobs of the same job name might exist. If you submit a sub job specified job name with --sparam "jnam=" option, the sub job is assumed to be associate with the latest step job.

For the submission order, the step number of a submitted job is incremented by one from 0, unless otherwise specified. The user can specify a step number in the --sparam "sn=" option.

```
$ pjsub --step --sparam "sn=10" stepjob1.sh   # The step number is set to 10.
```

When the user specifies a step number, the number must be greater than the largest step number at that point. You can check the step number by using the pjstat command. (See "2.4 Checking the Job Status.")

You can submit multiple sub jobs for one step job specified in arguments of pjsub command by delimiting with commas or blanks.

```
$ pjsub --step stepjob1.sh,stepjob2.sh      (or pjsub --step stepjob1.sh stepjob2.sh)
[INFO] PJM 0000 pjsub Job 300_0 submitted.
[INFO] PJM 0000 pjsub Job 300_1 submitted.
```

In the above example, the step number is set as 0, 1,... in the order of specification of the job scripts. If the --sparam "sn=" option is specified, the setting of step numbers begins with the specified value.
You can also submit multiple sub jobs for an already existing step job.

```
$ pjsub --step stepjob1.sh
[INFO] PJM 0000 pjsub Job 400_0 submitted.
$ pjsub --step --sparam "jid=400" stepjob2.sh,stepjob3.sh
 (or pjsub --step --sparam "jid=400" stepjob2.sh stepjob3.sh)
[INFO] PJM 0000 pjsub Job 400_1 submitted.
[INFO] PJM 0000 pjsub Job 400_2 submitted.
```

Note

Note the following about submitting multiple sub jobs.

- The option specified by the argument of the pjsub command is applied to all sub jobs that are submitted at the same time. Specify the option in the job scripts, if you want to specify different option for each job.
  Note the following when you specify the job name, the job ID and the step number.
  - If you want to set different job name for each sub job, specify the -N or --name option in each job script. When the --sparam "jnam=" option is specified in the job script, the --sparam "jnam=" option needs to be specified in all other scripts which are submitted at same time. Also their job names need to be same.
  - The option in the first job script is effective, if you specified the --sparam "jid=" options for specifying the job ID of step job in the plural job scripts which are submitted at same time.
  - The --sparam "sn=" option for specifying the step number specified in an argument of the pjsub command represents the step number for the first sub job. The step number of following sub jobs is incremented by one.
- If you want to specify the standard output file, the standard error output file or the job statistical information file for the sub job, these files need to be different file names for each sub job. For example, the step number is used for format of their file names. If these files are same name, the output of the latest sub job overwrites them.

Information

When the comma (,) is in the job script name, you need to specify "none" with the --script-delimiter option not to be interpreted as a separator.

```
$ pjsub --script-delimiter none --step step,job1.sh
```

In the submission of the second and subsequent sub jobs, use the --sparam "sd=" option to specify an operation based on the results of the preceding sub job. It is called the "dependency expression" of a step job.

The dependency expression is specified in the following format.

```
--sparam "sd=form[:[deletetype][:stepno[:stepno[...]]]]"
```

form represents a condition used to determine whether to execute a sub job that is submitted. *deletetype* represents the detailed operation performed if that sub job is not executed. The *form* condition applies to the execution results of a sub job, and *stepno* represents the step number of this sub job. If *stepno* is omitted, the results of the last sub job become the target.

The following tables list values that can be specified in *form* and *deletetype*.

form	Descriptions
NONE	Indicates that there is no dependent sub job. It is the same as not specifying "sd=". Each sub job that is submitted is executed when the preceding sub job ends. Note: Any subsequent sub job deleted by a preceding sub job will not be executed.
param==*value* *param*!=*value* *param*>*value* *param*<*value* *param*>=*value* *param*<=*value*	Indicates conditions for deleting (not executing) a sub job according to the *deletetype* specification. *param* is ec or pc. The meaning is as follows. ec: End status of the job script of a dependent sub job pc: Job end code (PJM code) of a dependent sub job You can specify multiple values for the conditions == and != by delimiting them with a comma (,).

Information

Difference between the job script end status (ec) and the job end code (PC)

Job end code (pc) shows whether the job operation management function had processed the job normally. Even if the exit status of the job script (ec) is not 0, the job end code is 0 when the job is processed normally.
The job end code might not be 0 though the exit status of the job script is 0 when the job could not be processed normally (ex. exceeding elapse time limit).
Users need to consider the dependency expression while taking account of the difference between the job end code (pc) and the job script exit status (ec). For the meanings of the values of the job end code (pc), see the description of the "PC" item in "Table A.1 Output items of the pjstat command" in "A.1 Output of the pjstat and pjstat -v."

deletetype	Description
one	Deletes only this sub job. The subsequent sub jobs that are dependent on the results of this sub job are not deleted. If *deletetype* is omitted, one is assumed specified.
after	Deletes this sub job and only the subsequent dependent sub jobs.
all	Deletes this sub job and all subsequent sub jobs.

The following example shows the submission of a sub job for a step job with job ID 500.
In this example, if the end code of the job script of the sub job with step number 0 is not 0, the subsequent sub jobs are not executed.

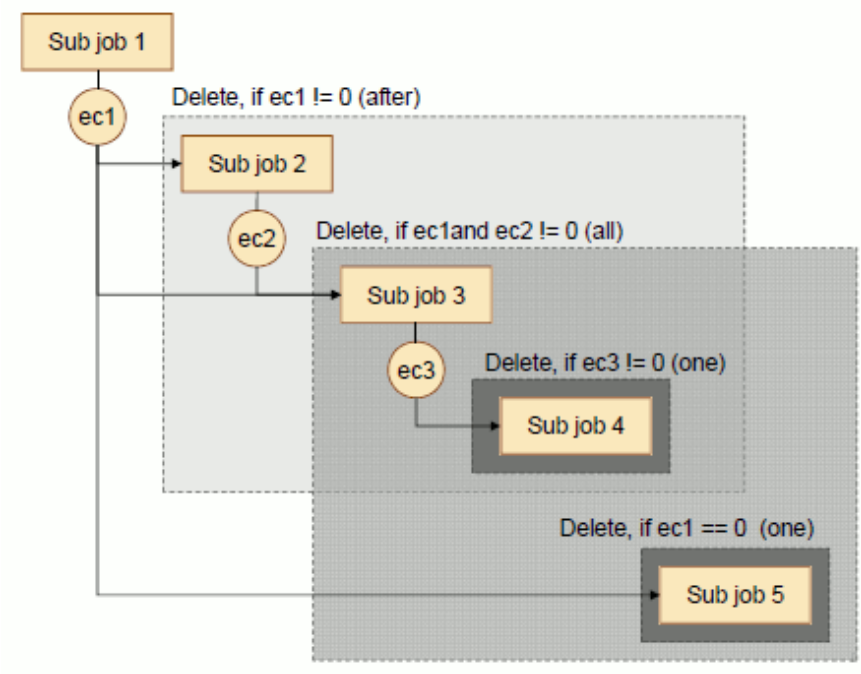
```
$ pjsub --step --sparam "jid=500,sd=ec!=0:all:0" stepjob2.sh
```

Note

Enclose the arguments of the --sparam option in single or double quotation marks so that they are recognized as a single character string.

The following is a specific example of a step job.

Figure 2.8 Example of a step job



- Sub job 1

  The sub job is executed first.

- Sub job 2

  If the end code of sub job 1 is not 0, sub job 2 and all other jobs dependent on sub job 2 are deleted. Sub jobs 3 and 4 are affected by the deletion of sub job 2, and they are also deleted.

- Sub job 3

  If the end codes of sub jobs 1 or 2 are not 0, sub job 3 and all subsequent jobs are deleted. Sub jobs 4 and 5 are also deleted.

- Sub job 4

  If the end code of sub job 3 is not 0, only sub job 4 is deleted. Sub job 5 is not deleted.

- Sub job 5

  If the end code of sub job 1 is 0, only sub job 5 is deleted.

The following shows this submitted step job.

```
$ pjsub --step --sparam "sn=1" job1.sh                                (1)
[INFO] PJM 0000 pjsub Job 500_1 submitted.                            (2)
$ pjsub --step --sparam "jid=500,sn=2,sd=ec!=0:after:1" job2.sh       (3)
$ pjsub --step --sparam "jid=500,sn=3,sd=ec!=0:all:1:2" job3.sh
$ pjsub --step --sparam "jid=500,sn=4,sd=ec!=0:one:3" job4.sh
$ pjsub --step --sparam "jid=500,sn=5,sd=ec==0:one:1" job5.sh
```

(1) Submitted as step number 1
(2) Job ID 500 in this example
(3) Submitting step number 2 and subsequent steps below

You can submit the sub jobs of a single step job to different resource units or resource groups. When doing this, you need to specify the names of the resource units or resource groups to which the sub jobs are to be submitted. If you omit this specification, the sub jobs will be submitted to the default resource unit or resource group that is set in the job ACL function.
The following example submits the sub jobs with step numbers 1 and 3 to the resource unit rscunitPG and the sub job with step number 2 to the resource unit rscunitFX:

```
$ pjsub --step -L "rscunit=rscunitPG" --sparam "sn=1" job-PG1.sh
[INFO] PJM 0000 pjsub Job 600_1 submitted.
$ pjsub --step -L "rscunit=rscunitFX" --sparam "jid=600,sn=2,sd=ec!=0:after:1" job-FX.sh
[INFO] PJM 0000 pjsub Job 600_2 submitted.
$ pjsub --step -L "rscunit=rscunitPG" --sparam "jid=600,sn=3,sd=ec!=0:all:1:2" job-PG2.sh
[INFO] PJM 0000 pjsub Job 600_3 submitted.
```

2.3.4.3 How to submit a workflow job

Workflow job is a method of controlling the submission of multiple jobs by a user. User creates the shell script that submits multiple jobs, and executes it as shown in "1.2.1.4 Workflow job."

2.3.4.4 How to submit a master-worker job

To submit a master-worker job, specify the --mswk option of the pjsub command.

```
$ pjsub --mswk job_script
```

Write the specification (-L, --rsc-list) of the upper limit value on resources, the node shape, etc. as required, in pjsub command arguments or a job script.

Note

- Master-worker jobs do not support a rank number directory. Therefore, the --mpi use-rankdir option of the pjsub command is ignored even if specified at the master-worker job submission time.
- The parallel execution environment of the job operation software determines the node that will generate a worker process in a master-worker job. Alternatively, the user specifies the node when generating the process.
  Therefore, specification of the --mpi rank-map-hostfile option of the pjsub command has no meaning. This option is ignored even if specified.

2.3.4.5 How to submit an interactive job

Specify the --interact option of the pjsub command to declare the job as an interactive job.

Information

Administrators who can log in to the compute cluster management node can submit jobs by using the pjsub command from the node. However, they must submit only interactive jobs from the login node.

With an interactive job, the user can interactively enter the job contents from a terminal. Alternatively, the user can specify the job contents in the job script in the same way as for a batch job.

If no job script is specified in the arguments of the pjsub command, interactive input of the job contents is assumed, and the shell is started on pseudo terminal and waits for input.

```
$ pjsub --interact
[INFO] PJM 0000 pjsub Job 405916 submitted.             (1)
[INFO] PJM 0081 .connected.                             (2)
[INFO] PJM 0082 pjsub Interactive job 405916 started.   (3)
$                                                       (4)
...
```

```
$ exit                                                    (5)
[INFO] PJM 0083 pjsub Interactive job 405916 completed. (6)
```

(1) Message indicating interactive job submission
(2) Message indicating that the interactive job is being prepared
(3) Message indicating the start of the interactive job
(4) Shell prompt in the interactive job
(5) Exit from the shell
(6) Message indicating the end of the interactive job

The prompt in this example is that displayed by the shell on the compute node that executes the interactive job. The path of the current directory on the compute node is the same as that of the current directory on the login node when the pjsub command was executed.

The interactive job ends upon exit from the shell or when the job is deleted by the pjdel command. (See "2.5.1 Deleting a job.")

The input from a pseudo terminal is passed to the shell in the interactive job. To direct the special operation to the interactive job, use a tilde sign '~' as an escape character. The following operation can be directed by the tilde sign and the input following it. The tilde sign as the escape character should be the first character in the new line.

Table 2.25 Escape characters in the interactive job

Input	Action
~~	Sends the tilde sign to the shell
~.	Terminates the interactive job. The shell that was executed in the interactive job is terminated, and the pseudo terminal is closed.
~<Ctrl+z>	Suspend the pjsub command that executes interactive job, and back to the pseudo terminal in which the pjsub command is executed.
~?	Display the list of the escape characters.
Key sequence <Ctrl+\>, <Ctrl+_>, <Ctrl+\>, <Ctrl+_>, <Q(shift+q)>	In interactive jobs of McKernel mode, entering this key sequence will terminate the job.

If the job script is specified in the arguments of the pjsub command, the job is executed according to the job script contents in the same way as a batch job.

```
$ pjsub --interact interactjob.sh
[INFO] PJM 0000 pjsub Job 405918 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 405918 started.
...                                          <- Job script output contents
[INFO] PJM 0083 pjsub Interactive job 405918 completed.
```

Immediately after the interactive job is submitted, computer resources are allocated, and the job is executed. If the job cannot be executed immediately because of insufficient resources, it waits for allocation of resources. This may affect the execution of subsequent jobs.

To avoid this issue, you can specify a maximum wait time (seconds) for resource allocation for an interactive job by using the --sparam "wait-time=" option. If the resource allocation wait time exceeds the specified time, the job is canceled.

```
$ pjsub --interact --sparam "wait-time=600"                                    (1)
[INFO] PJM 0000 pjsub Job 291 submitted.
[INFO] PJM 0081 .
[INFO] PJM 0080 pjsub Interactive job 291 is canceled due to the resource allocation timeout.
       The timeout period "t" can be specified by "--sparam wait-time=t".      (2)
```

(1) The maximum wait for resource allocation is 600 seconds (10 minutes).
(2) The job was canceled because the wait time exceeded 600 seconds.

The following table lists the pjsub command options that are ignored in an interactive job.

Table 2.26 pjsub command options ignored in an interactive job

Option	Description
--at	Job execution start time
-e, --err	Standard error output file
-j	Directing the standard error output to the standard output file
--bulk, --step	Job model
-m e	E-mail notification of execution end
-m r	E-mail notification of execution restart
--restart	Enable automatically re-execution of a job
-o, --out	Standard output file
-p	Job priority
-w	Wait for the pjsub command to return

Note

When running an interactive job, if you exit the interactive job after entering <Ctrl+c>, you will see the following message "[INFO] PLE 0094" before the normal exit message, which should be safely ignored:

```
[INFO] PLE 0094 plexec The interactive job has received the signal.(sig=2)
[INFO] PJM 0083 pjsub Interactive job 405920 completed.
```

2.3.5 Specifying a node selection policy [PG]

On PRIMERGY servers, you can specify a node selection policy in addition to node resources. (See "1.6.4 Allocation in units of virtual nodes.")

2.3.5.1 Virtual node placement policy

You can specify a virtual node placement policy for only a virtual node allocated job. To specify one, use the -P "vn-policy=" option of the pjsub command.

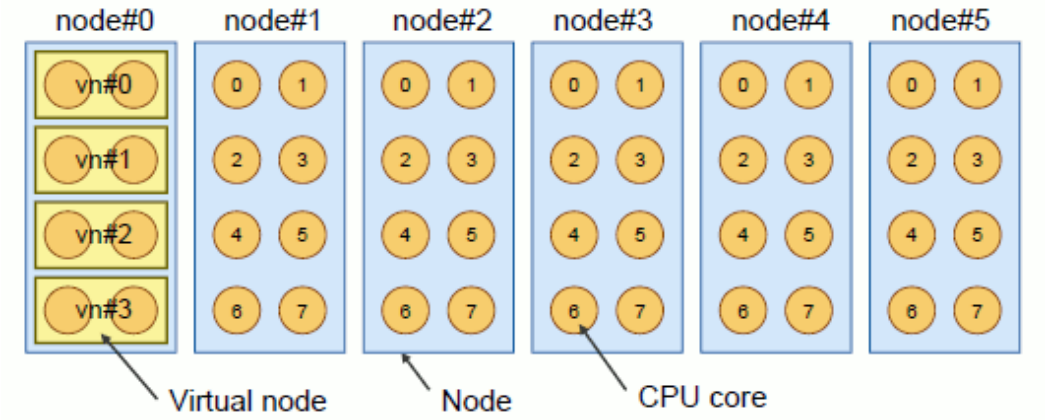
Absolutely PACK

To always place all virtual nodes on one node, specify the -P "vn-policy=abs-pack" option.

The following example shows placement with Absolutely PACK of four virtual nodes having two CPU cores.

```
$ pjsub -L "vnode=4,vnode-core=2" -P "vn-policy=abs-pack" job.sh
or
$ pjsub -L "vnode=4(core=2)" -P "vn-policy=abs-pack" job.sh
```

Figure 2.9 Virtual node placement with Absolutely PACK



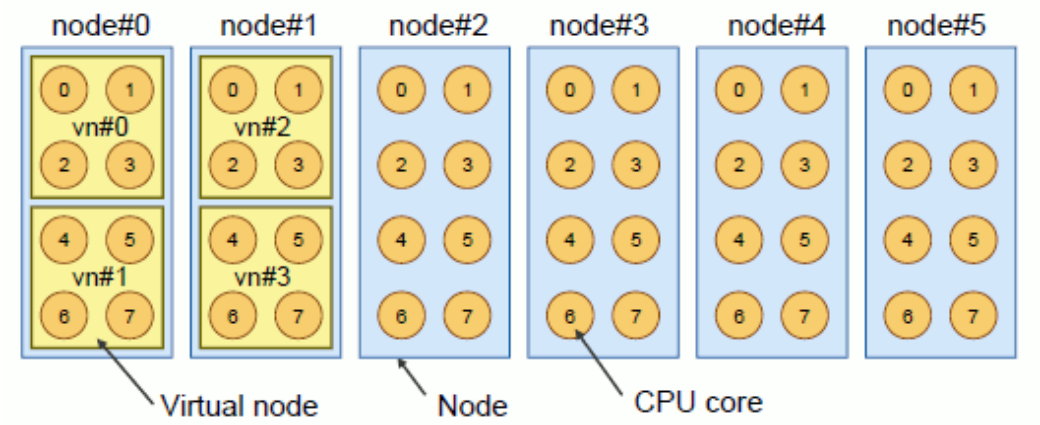
PACK

To place virtual nodes on only one node if at all possible, specify the -P "vn-policy=pack" option.

The following example shows placement with PACK of four virtual nodes having four CPU cores.

```
$ pjsub -L "vnode=4,vnode-core=4" -P "vn-policy=pack" job.sh
or
$ pjsub -L "vnode=4(core=4)" -P "vn-policy=pack" job.sh
```

Figure 2.10 Virtual node placement with PACK



In the above example, two nodes are used because only two virtual nodes can be placed on one node.

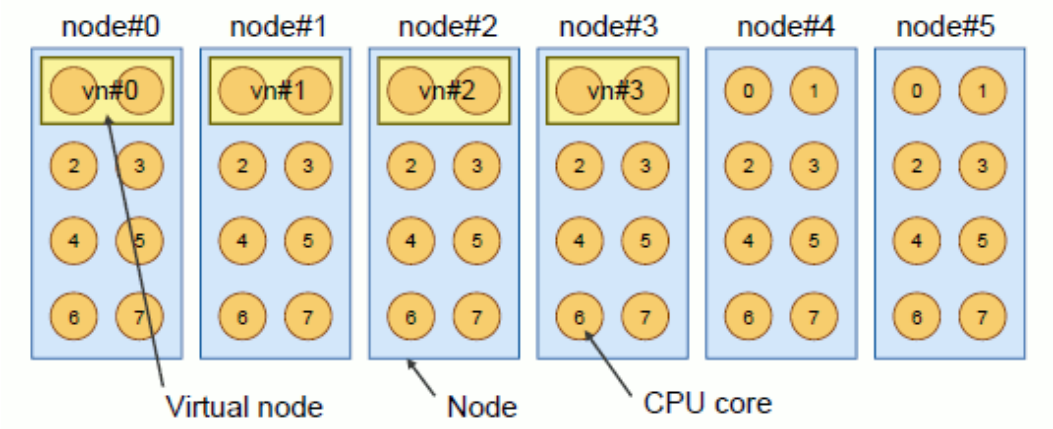
Absolutely UNPACK

To always place each virtual node on different nodes, specify the -P "vn-policy=abs-unpack" option.

The following example shows placement with Absolutely UNPACK of four virtual nodes having two CPU cores.

```
$ pjsub -L "vnode=4,vnode-core=2" -P "vn-policy=abs-unpack" job.sh
or
$ pjsub -L "vnode=4(core=2)" -P "vn-policy=abs-unpack" job.sh
```

Figure 2.11 Virtual node placement with Absolutely UNPACK

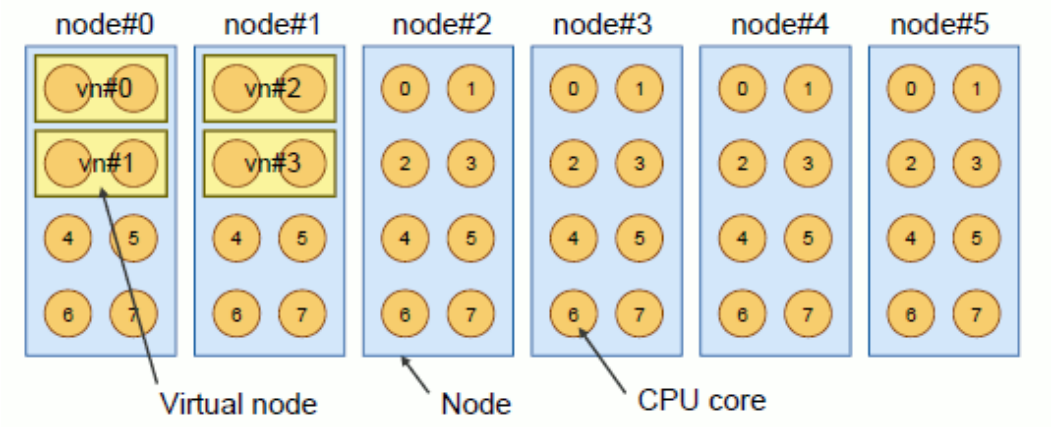


With Absolutely UNPACK, you can also define a specification where *N* (number) virtual nodes are always placed on different nodes. In this case, specify the -P "vn-policy=abs-unpack=*N*" option.

The following example shows placement with Absolutely UNPACK of four virtual nodes having two CPU cores, with two virtual nodes per node.

```
$ pjsub -L "vnode=4,vnode-core=2" -P "vn-policy=abs-unpack=2" job.sh
or
$ pjsub -L "vnode=4(core=2)" -P "vn-policy=abs-unpack=2" job.sh
```

Figure 2.12 Placement of two virtual nodes per node with Absolutely UNPACK



Note

The pjsub command with "vnode=*M*" (the number of virtual nodes) and "vn-policy=abs-unpack=*N*" specified returns an error if *M* is not a multiple of *N*.

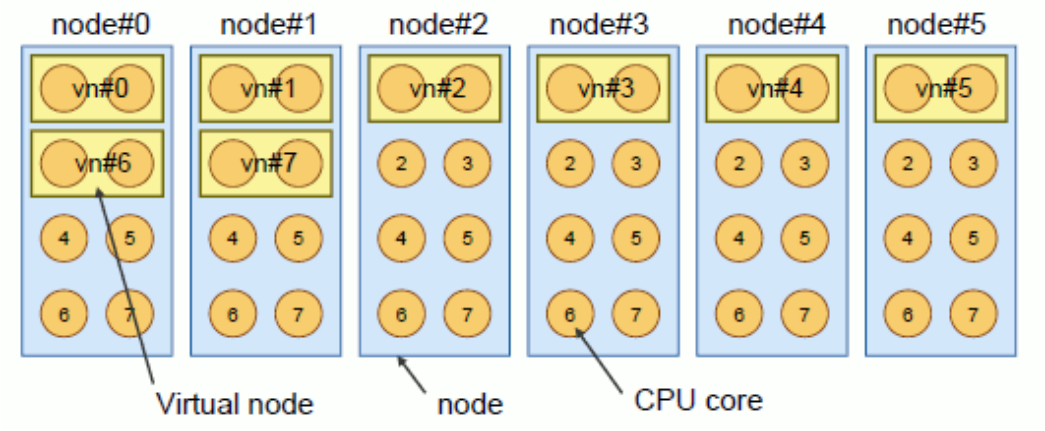
UNPACK

To place each virtual node on only different nodes if at all possible, specify the -P "vn-policy=unpack" option.

The following example shows placement with UNPACK of eight virtual nodes having two CPU cores.

```
$ pjsub -L "vnode=8,vnode-core=2" -P "vn-policy=unpack" job.sh
or
$ pjsub -L "vnode=8(core=2)" -P "vn-policy=unpack" job.sh
```

Figure 2.13 Virtual node placement with UNPACK

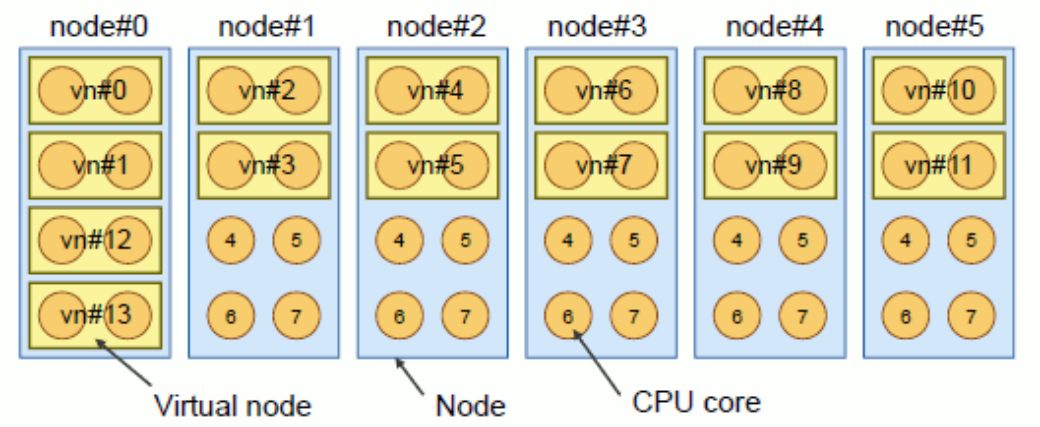


With UNPACK, you can also define a specification where N (number) virtual nodes are placed on only different nodes if at all possible. In this case, specify the option -P "vn-policy=unpack=*N*".

The following example shows placement with UNPACK of 14 virtual nodes having two CPU cores, with only two virtual nodes per node if at all possible.

```
$ pjsub -L "vnode=14,vnode-core=2" -P "vn-policy=unpack=2" job.sh
or
$ pjsub -L "vnode=14(core=2)" -P "vn-policy=unpack=2" job.sh
```

Figure 2.14 Placement of two virtual nodes per node with UNPACK



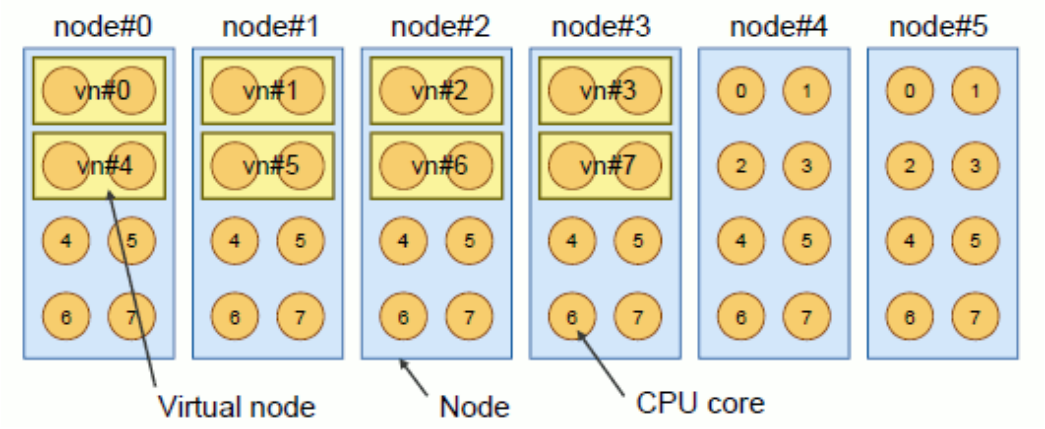
Note

The pjsub command with "vnode=*M*" (the number of virtual nodes) and "vn-policy=unpack=*N*" specified returns an error if *M* is not a multiple of *N*.

Suppose you specify -P "vn-policy=unpack" with not only a number of virtual nodes but also a number of nodes (-L vnode=M,node=*N*, and *M* >= *N*). Then, *M*/*N* virtual nodes will be placed on each node.

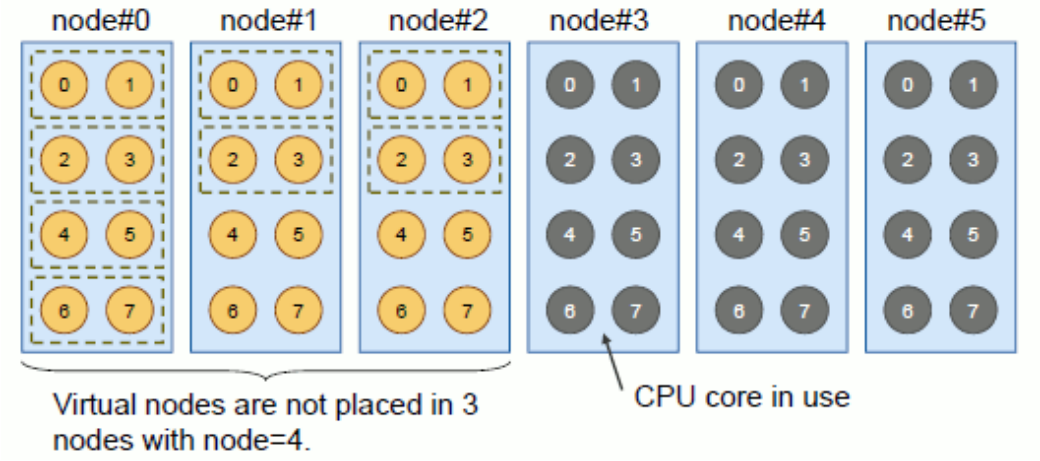
```
$ pjsub -L "vnode=8,vnode-core=2,node=4" -P "vn-policy=unpack" job.sh
or
$ pjsub -L "vnode=8(core=2),node=4" -P "vn-policy=unpack" job.sh
```

Figure 2.15 UNPACK and specification of the number of nodes (1)



Also, if the number of nodes is specified, the specified number of nodes is always necessary. Therefore, if there are insufficient nodes as shown in following figure, virtual nodes cannot be placed even when -P "vn-policy=unpack" is specified.

Figure 2.16 UNPACK and specification of the number of nodes (2)



In the above example, UNPACK placement does not use three nodes only (nodes 0, 1, 2) since the specified number of nodes is four.

2.3.5.2 Rank map

For details on rank maps, see "2.3.6 Submitting an MPI job"

2.3.5.3 Node selection method

There are two methods of selecting the nodes for placing virtual nodes: distributed method and concentrated method.

The administrator sets which method is used, and users cannot specify it.

The following shows the difference in allocation between the distributed method and concentrated method. In this case, five virtual nodes having two CPU cores are submitted in a way (UNPACK) that they are placed on only different nodes if at all possible.

```
$ pjsub -L "vnode=5,vnode-core=2" -P "vn-policy=unpack" job.sh
or
$ pjsub -L "vnode=5(core=2)" -P "vn-policy=unpack" job.sh
```

Figure 2.17 Node selection with the distributed method

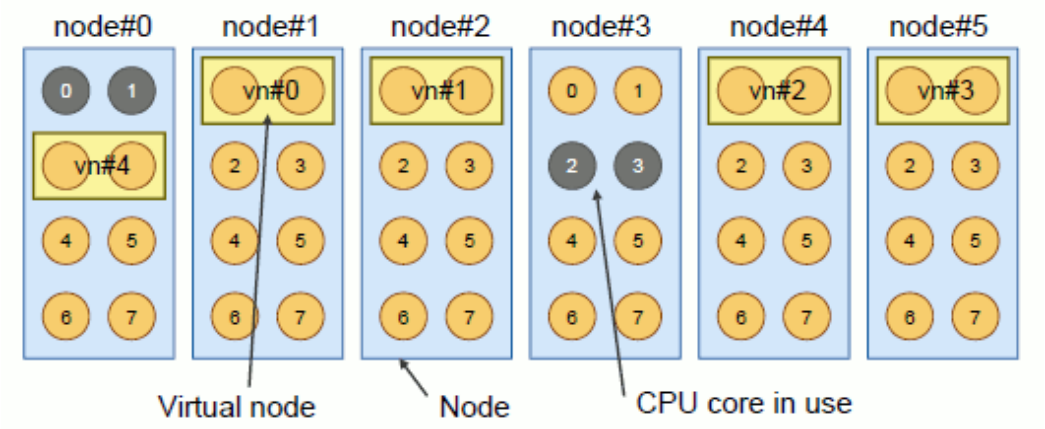
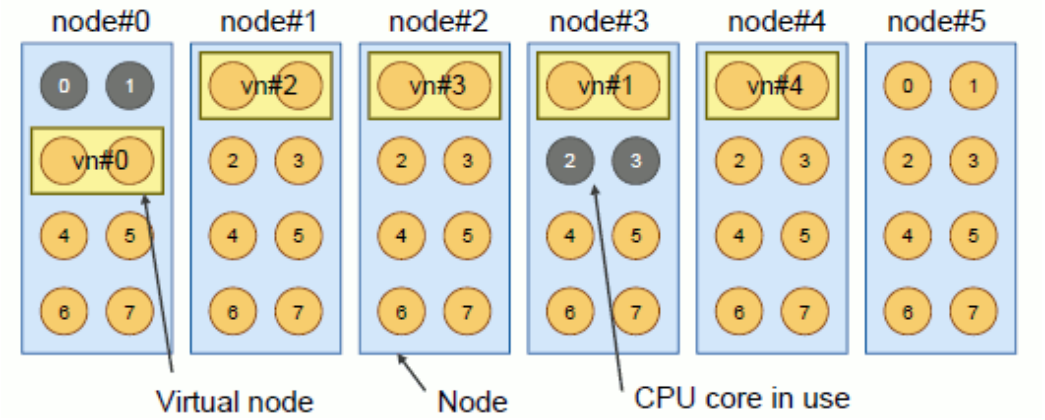


Figure 2.18 Node selection with the concentrated method



2.3.5.4 Priority control of allocated nodes

Priority control of allocated nodes is a method of selecting nodes according to the priority set for the nodes.

The administrator sets whether to use this method and the priority of the nodes. Therefore, users do not have to be aware of either when submitting jobs.

2.3.5.5 Execution mode policy

An execution mode policy specifies whether to allow submitted virtual node allocated jobs to share nodes with other jobs.

For this policy, SIMPLEX (occupy) and SHARE (share) can be specified in the -P "exec-policy=" option of the pjsub command.

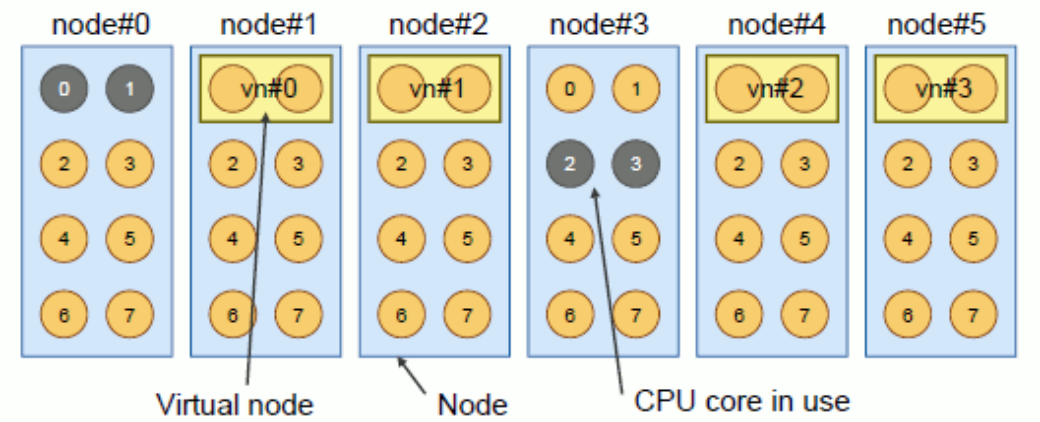
The following examples specify SIMPLEX and SHARE respectively.

SIMPLEX

In this example, four virtual nodes having two CPU cores are always placed on different nodes, and the job occupies these nodes.

```
$ pjsub -L "vnode=4,vnode-core=2" -P "vn-policy=abs-unpack,exec-policy=simplex" job.sh
or
$ pjsub -L "vnode=4(core=2)" -P "vn-policy=abs-unpack,exec-policy=simplex" job.sh
```

Figure 2.19 Node occupation with SIMPLEX specification



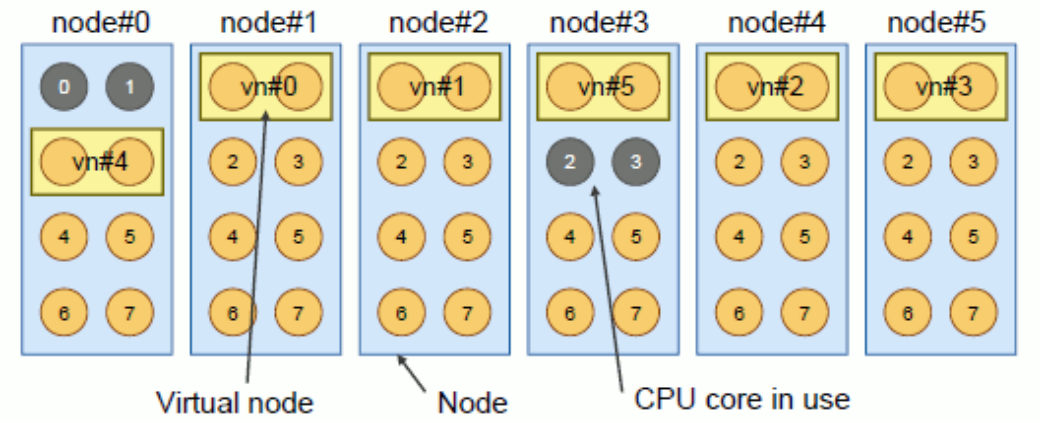
In the above example, only four nodes (node#1, node#2, node#4, and node#5) can be occupied. Therefore, five or more virtual nodes cannot be allocated with SIMPLEX.

SHARE

In this example, six virtual nodes having two CPU cores are always placed on different nodes, and the nodes are allowed to be shared with other jobs.

```
$ pjsub -L "vnode=6,vnode-core=2" -P "vn-policy=abs-unpack,exec-policy=share" job.sh
or
$ pjsub -L "vnode=6(core=2)" -P "vn-policy=abs-unpack,exec-policy=share" job.sh
```

Figure 2.20 Node sharing with SHARE specification



The fifth and sixth virtual nodes (vn4, vn5) in the above example share the nodes with other jobs.

2.3.6 Submitting an MPI job

When submitting an MPI job, you can specify the following parameters in the --mpi option of the pjsub command:

- Shape of the process [FX]
- Number of processes to create
- Rules on assigning ranks for created processes

The following example describes specification of the --mpi option.

Note

- To execute an MPI program on a single node (node = 1) or a single virtual node (vnode=1), be sure to specify the --mpi option.

- This section describes how to submit, as a job, an MPI program created in Development Studio. For details on how to execute programs of an MPI processing system other than Development Studio, see "Appendix C Executing programs of MPI processing system other than Development Studio."

2.3.6.1 Specifying the shape of the process [FX]

You can specify the shape of the initial processes (node shape) by using the shape parameter of the --mpi option of the pjsub command for MPI job as a node allocated job on the FX servers.

The types of process shape are one-dimensional, two-dimensional, and three-dimensional. The specified number of dimensions must be the same as the node shape specified in the node parameter of the -L option (or --rsc-list). If the shape parameter is omitted from the --mpi option, the specification in the node parameter of the -L option is used.

```
[one-dimensional shape]   --mpi "shape=X"
[two-dimensional shape]   --mpi "shape=XxY"
[three-dimensional shape] --mpi "shape=XxYxZ"
```

The parallel execution environment selects nodes so that the processes generated at MPI program startup and the dynamically generated processes are not allocated to the same nodes. Therefore, node shape (-L node=) allocated to the MPI job should be an enough size where all the MPI processes including the process dynamically created can be executed at the same time in the job.

Information

If you specify the correspondence between allocated nodes and processes (ranks) by the following means, you can allocate the MPI processes generated at startup and the dynamically created MPI processes to the same nodes:

- --vcoordfile option of the mpiexec command
- info key vcoordfile of the MPI_Comm_spawn function and MPI_Comm_spawn_multiple function

For details on the mpiexec command, the MPI_Comm_spawn function, and the MPI_Comm_spawn_multiple function, see the "MPI User's Guide," which is a Development Studio manual.

For instance, dynamic process creation fails because it is allocated only by one node when '-L node=1' is specified at submitting the MPI job. When the process dynamically created ends, the node that it used can be used to create the process dynamically newly.

The following example shows specification of a three-dimensional process shape (four nodes on the X-axis, three on the Y-axis, and two on the Z-axis) for the MPI program prog_A.

```
$ cat job.sh                                                    (1)
#!/bin/sh
...
mpiexec ./prog_A
$ pjsub -L "node=4x3x2" --mpi "shape=4x3x2" job.sh              (2)
```

(1) job.sh job script that executes the MPI program prog_A
(2) Submits a job with a specified node shape of 4 x 3 x 2 and process shape 4 x 3 x 2.

2.3.6.2 Specifying the number of processes to create

The number of processes created at starting the MPI program or can be created dynamically are specified by the proc parameter of --mpi option of the pjsub command.

```
--mpi "proc=procnum"
```

The value of proc parameter is as follows.

- For a job to be executed on the FX servers

  The maximum value that can be specified for the proc parameter is "<*shape parameter*>* *N*". Here, *N* is the number of CPU cores per computer node.
  Specifying the maximum value makes the job a flat parallel job, which executes processes on all the cores of each node so that the

processes are fixed to different cores without duplication.
In other words, this maximum value is as follows: the number of processes generable in a node is "*<proc parameter>* / *<The number of nodes specified with "shape" parameter>*" (It rounds up below the decimal point). This value is also applied to the processes which are created dynamically.
When the proc parameter is omitted, the number of nodes specified with the shape parameter is used. When the shape parameter is also omitted, the number of nodes specified with the node parameter of -L option is used

- For a job to be executed on the PRIMERGY servers

  The number of processes that can be generated on a virtual node is 1. If the node parameter specifies nodes, the number of processes that can be generated on a node is also 1 since one node is handled in the same way as one virtual node.
  Therefore, the maximum value that can be specified for the proc parameter is equal to the number of nodes or virtual nodes as specified.
  If the proc parameter is omitted, the specified number of nodes or virtual nodes is used.

Note

If the specified value exceeds the maximum value for the proc parameter, the operation is as follows.

- For a job executed on the FX servers
  The pjsub command rejects the acceptance of the job.

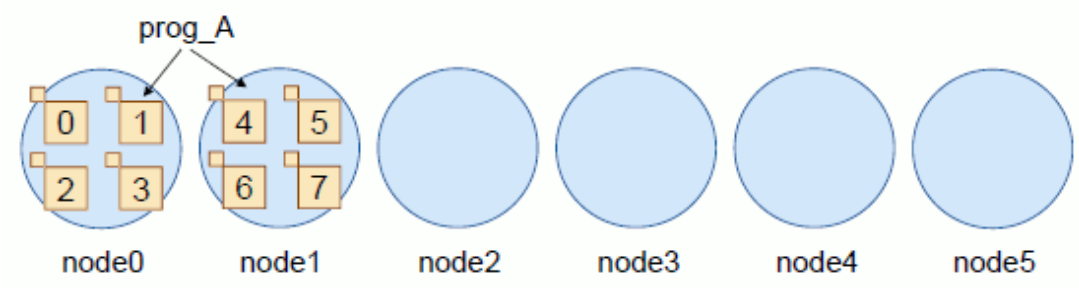
- For a job executed on the PRIMERGY servers
  An error message of the internal command plexec of the parallel execution environment is output to the standard error output file of the job during the execution of the job. For details of the message, see "plexec command" in "Chapter 3 Command Reference for End-user" of the "Job Operation Software Command Reference" manual.

```
[ERR.] PLE 0070 plexec The number of processes "n" must be smaller than or equal to the number of virtual nodes "m": "pjsub -L vnode=m", "pjsub --mpi proc=n".
```

The following is an example of MPI program that creates processes dynamically, and it explains the relation among node, shape and proc parameter.

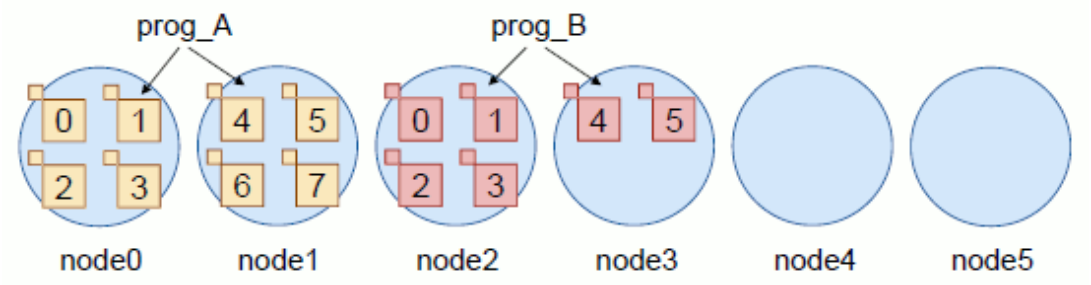
```
$ cat job.sh
#!/bin/sh
...
mpiexec ./prog_A
$ pjsub -L "node=6" --mpi "shape=2,proc=8" job.sh
```

1. In this example, 6 nodes are allocated to a node allocated job (-L "node=6")
2. When the MPI program prog_A is executed, 8 processes are created on two nodes, node0 and node1 ("shape=2,proc=8"). Four processes are created in a node (proc/shape=4).



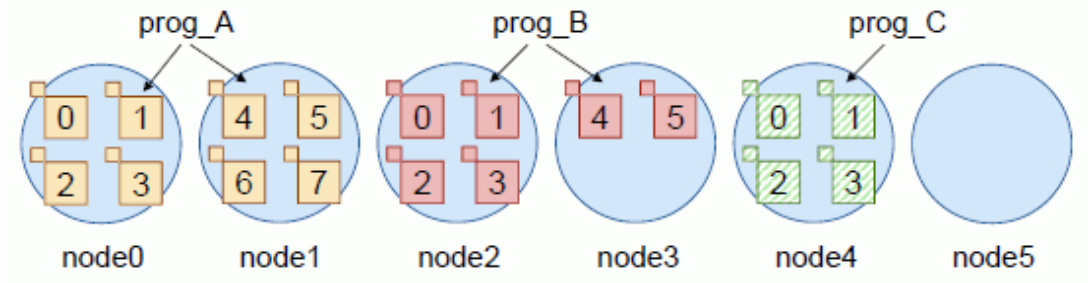
3. When the MPI program prog_A creates 6 processes of the MPI program prog_B using function MPI_Comm_spawn, the processes are created on node2 and node3 ("shape=2"). The maximum number of processes per a node is 4 as well as prog_A.

```
MPI_Comm_spawn("prog_B", NULL, 6, ...);
```



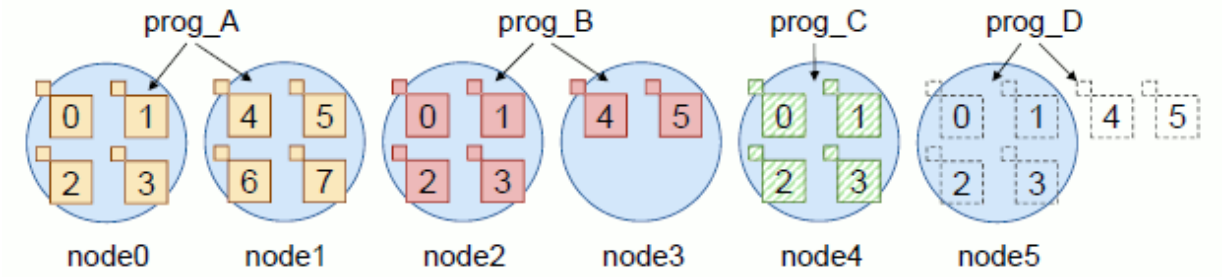
4. When the MPI program prog_A creates 4 processes of program prog_C, the processes are created on node4 as well as prog_A. At this time, the processes are created on node 4 so that prog_C should not share nodes with prog_B though there are free CPUs in node 3.

```
MPI_Comm_spawn("prog_C", NULL, 4, ...);
```



5. The generation of process fails, when the MPI program prog_A try to create 6 processes of program prog_D further. Because free node is only node5 though the program prog_D needs 2 nodes.

```
MPI_Comm_spawn("prog_D", NULL, 6, ...);
```



Note

This example shows the case where the --mpi shape parameter of pjsub is specified. However, even without specifying the parameter, the program generates dynamic processes on a different node than the nodes that have the static processes. (The parameter may be omitted to generate dynamic processes of an MPI job without specifying info key vcoordfile of the MPI_Comm_spawn function or MPI_Comm_spawn_multiple function.)

For a node allocated job, you can also specify an upper limit on the number of MPI processes generated per node.

Table 2.27 Upper limit on the number of MPI processes generated per node

Option	Meaning
--mpi max-proc-per-node=*n*	Upper limit for number of MPI processes generated per node. This is valid only for node allocated jobs.

When this is specified with the --mpi proc=*n* option, the operation is as follows.

Table 2.28 Combinations of options for specifying the number of MPI processes generated and their operations

--mpi proc=*n1*	--mpi max-proc-per-node=*n2*	Operation
With specification	Without specification	[At start of program] In accordance with the rank allocation rule in the next section, *n1*/*shape* or *n1*/*node* process (rounded up after the decimal point) MPI processes are generated per node, up to the total number of *n1* processes. [At generation of dynamic process] Up to *n1*/*shape* or *n1*/*node* MPI processes (rounded up after the decimal point) in one node can be generated. *shape*: Number of nodes specified with the -L shape parameter (only for FX server) *node*: Number of nodes specified with the -L node parameter (when not specifying -L shape parameter)
Without specification	With specification	[At start of program] *n2* MPI processes for each node are generated. [At generation of virtual processes] Up to *n2* MPI processes can be generated for each node.
With specification	With specification	[At start of program] In accordance with the rank allocation rule in the next section, up to *n2* MPI processes are generated for one node, up to the total number of *n1* processes. [At generation of dynamic process] Up to *n2* MPI processes can be generated for each node.

Note

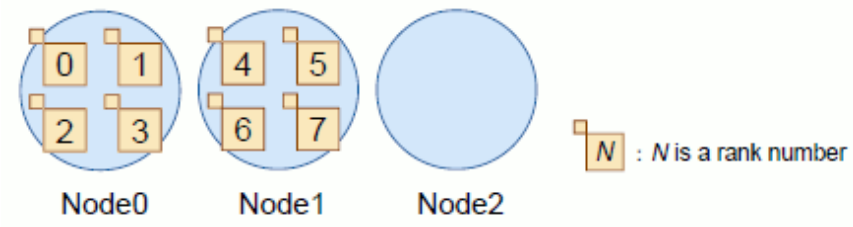
The above operations are the default operations that result when an option is not specified for the mpiexec command that is specified within the job script.

Examples are given below.

[Example 1]

```
[Job Script]
#PJM -L node=3
#PJM --mpi "shape=2,proc=8"
mpiexec a.out
```

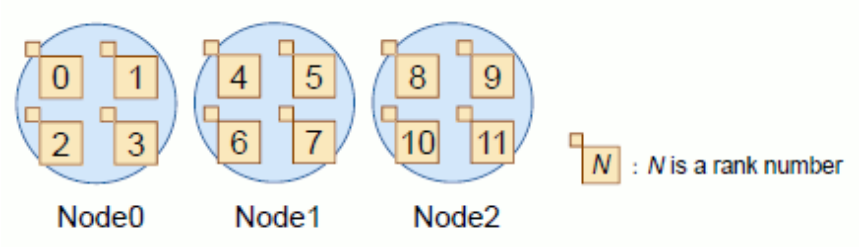
In this case, the MPI process placement is as follows.



[Example 2]

```
[Job Script]
#PJM -L node=3
#PJM --mpi max-proc-per-node=4
mpiexec a.out
```

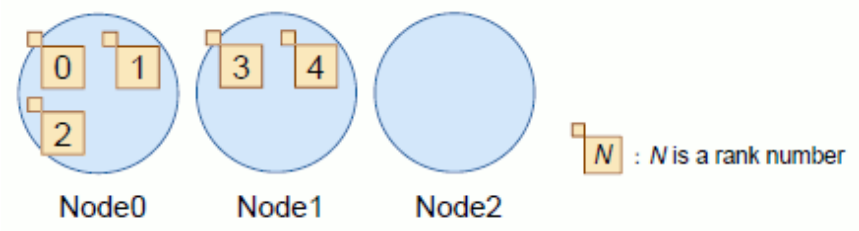
In this case, the MPI process placement is as follows.



[Example 3]

```
[Job Script]
#PJM -L node=3
#PJM --mpi proc=5
#PJM --mpi max-proc-per-node=3
mpiexec a.out
```

In this case, the MPI process placement is as follows.



2.3.6.3 The rules on assigning nodes for the ranks

MPI assigns the number that is called rank to each process for the identification of the one like process ID. The job operation management function assigns nodes for each process in order of the rank. You can specify a rule on selecting the nodes to be assigned for ranks in the --mpi option of the pjsub command.

- To assign nodes in sequence from the same node for consecutive ranks, specify the rank-map-bychip parameter. For details, see "2.3.6.4 rank-map-bychip parameter."
  To assign different nodes in sequence for consecutive ranks, specify the rank-map-bynode parameter. For details, see "2.3.6.5 rank-map-bynode parameter."
  These parameters are mutually exclusive.
  If neither is specified, nodes are assigned as follows.
  - For FX servers (only for node allocated jobs)
    It operates assuming that the rank-map-bychip parameter is specified.
  - For PRIMERGY servers (only for virtual node allocated jobs)
    Virtual node IDs are set in the order of virtual node placement based on the virtual node placement policy. On PRIMERGY servers, virtual node IDs correspond to the rank order (rank numbers).

Information

To generate one rank (process) for one node, the node assignment method is the same regardless of the specified parameter.

- For FX servers, you can specify which nodes are selected and their sequence in the rank-map-hostfile parameter when assigning different nodes for ranks. For details, see "2.3.6.7 Node specification by rank-map-hostfile parameter [FX]."
  You can specify the rank-map-hostfile parameter together with the rank-map-bychip or rank-map-bynode parameter.
  If the rank-map-hostfile parameter specification is omitted, the rank-map-bynode or rank-map-bychip parameter is assumed to be *rankmap*. For details, see "2.3.6.6 The order of assigning node specified for rankmap [FX]."

For the FX servers, the efficiency of the communication can be improved by arranging it so that the communication distance is shorten between the processes.

Note

- For FX servers, you can specify the rank-map-bychip, rank-map-bynode, and rank-map-hostfile parameters only for node allocated jobs.
- Allocating PRIMERGY servers in units of nodes is like allocating virtual nodes that are the same size as nodes. Therefore, there is no meaning to specifying the rank-map-bychip and rank-map-bynode parameters for this allocation.

It explains the assigning rule that can be specified --mpi option at the following.

2.3.6.4 rank-map-bychip parameter

This section describes the procedure for allocating nodes with the rank-map-bychip parameter.

For FX servers (only for node allocated jobs)

The followings are the formats of rank-map-bychip parameter.

```
--mpi "rank-map-bychip[:rankmap]"
--mpi "rank-map-bychip,rank-map-hostfile=filename"
```

Nodes are allocated as follows:

1. Allocate the specified number of processes for one node.
   For this number of ranks, specify the value of (proc parameter)/(number of nodes in shape parameter (rounded up to a whole number)). If the shape parameter is not specified, the specified value in the node parameter is used.
2. Select the next node.
   You can specify the order of nodes in the *rankmap* specification or rank-map-hostfile parameter. (See "2.3.6.7 Node specification by rank-map-hostfile parameter [FX].") If *rankmap* and the rank-map-hostfile parameter are specified at the same time, the rank-map-hostfile parameter has priority. For details on values specified in *rankmap*, see "2.3.6.6 The order of assigning node specified for rankmap [FX]."
3. Repeat steps 1 and 2 to assign nodes for all ranks.

For PRIMERGY servers (only for virtual node allocated jobs)

The following is the format of rank-map-bychip parameter.

```
--mpi "rank-map-bychip=n"
```

Virtual nodes are allocated as follows:

1. Place *n* virtual nodes, which is the number specified in --mpi "rank-map-bychip=*n*", on one node.
   Specify the rank-map-bychip=*n* parameter together with unpack=*m* or abs-unpack=*m*, where *m* must be a multiple of *n*.
   The order of the virtual nodes to allocate is ascending order of node ID and virtual node ID. Also, the rank number of the process to allocate corresponds to the virtual node ID.
2. Select the next node.
3. Repeat steps 1 and 2 until all the virtual nodes are allocated.

2.3.6.5 rank-map-bynode parameter

This section describes the procedure for allocating nodes with the rank-map-bynode parameter.

For FX servers (only for node allocated jobs)

The followings are the formats of rank-map-bynode parameter.

```
--mpi "rank-map-bynode[=rankmap]"
--mpi "rank-map-bynode,rank-map-hostfile=filename"
```

Nodes are allocated as follows:

1. Once a node is assigned for one rank, assign another node for the next rank.
   The order of the assigned node can be specified by *rankmap* or the rank-map-hostfile parameter (See "2.3.6.7 Node specification by rank-map-hostfile parameter [FX]"). It gives priority to the rank-map-hostfile parameter when *rankmap* and the rank-map-hostfile parameter are specified at the same time. See "2.3.6.6 The order of assigning node specified for rankmap [FX]" for the specification for *rankmap*.
2. After finishing node assignment for all ranks, return to the first assigned node.
3. Repeat steps 1 and 2 to assign nodes for all ranks.

For PRIMERGY servers (only for virtual node allocated jobs)

The followings are the formats of rank-map-bynode parameter.

```
--mpi "rank-map-bynode"
```

Virtual nodes are allocated as follows:

1. Once a virtual node in a node is assigned for one rank, assign a virtual node in another node for the next rank.
   The order of the virtual nodes to allocate is ascending order of node ID and virtual node ID. Also, the rank number of the process to allocate corresponds to the virtual node ID.
2. After finishing allocation for the virtual nodes in all the allocated nodes, return to the node allocated first, and allocate the another unused virtual node to ranks.
3. Repeat steps 1 and 2 to assign virtual nodes for all ranks.

2.3.6.6 The order of assigning node specified for rankmap [FX]

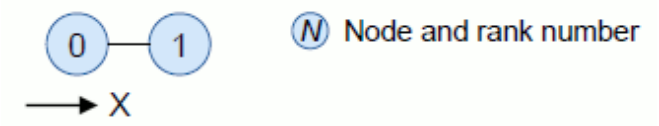
The *rankmap* specified with rank-map-bynode or the rank-map-bychip parameter for the node allocated job on FX servers is the one that which axially whether the node is assigned to the process and specified the node shape by the shape parameter. Specification is expressed by combining character X, Y, and Z that shows the axis. For example, you appoint it with "XY" in the case of the two-dimensional shape and appoint it like "XYZ" in the case of the three-dimensional shape. When the coordinate origin is assumed to be rank 0, the rank is arranged axially of the specification with *rankmap*, and it reaches the edge of shape, it moves to the following axis.

The *rankmap* cannot be specified for one dimension shape. In that case, the node shape is considered to be a torus X axially. The assigning node moves from the node of rank 0 to the adjoining node.

The image of the node assigned when one dimension shape is specified is shown as follows.

```
[one dimension shape] --mpi "rank-map-bynode"
[one dimension shape] --mpi "rank-map-bychip"
```

Figure 2.21 The order of assigning node in one dimension



The assigning node is done for the job that specifies the node shape by one dimension according to one dimension coordinates.

When two is specified for the shape parameter of --mpi option and four is specified for the proc parameter, each process assigning of the rank-map-bynode parameter and the rank-map-bychip parameter becomes it as follows.

Table 2.29 When you specify "--mpi shape=2 proc=4" by assigning one dimension the node

The order of assigning node	Assigned rank	
	For rank-map-bynode	For rank-map-bychip
0th	rank 0 and 2	rank 0 and 1
1st	rank 1 and 3	rank 2 and 3

Figure 2.22 For rank-map-bynode (one dimension)

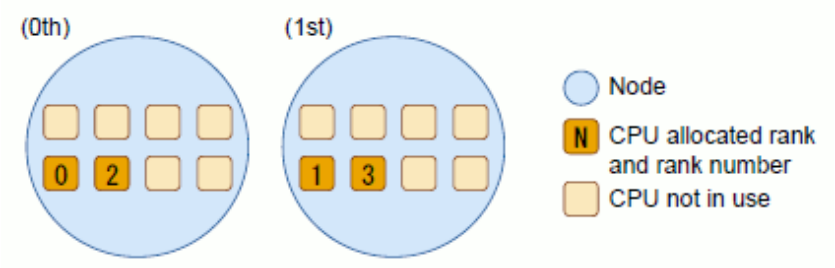
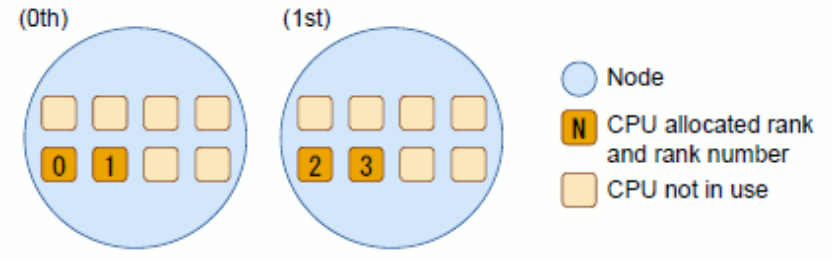


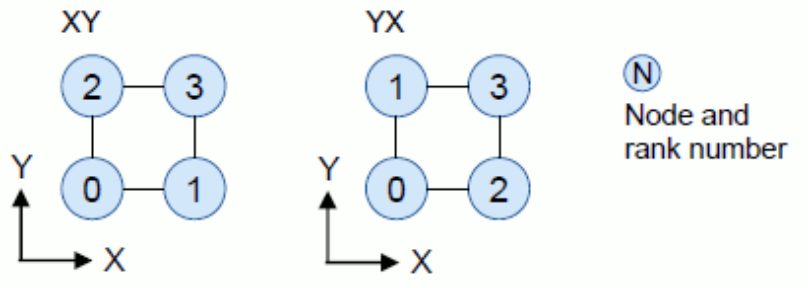
Figure 2.23 For rank-map-bychip (one dimension)



Next, the image of the node assigned when two dimensions shape is specified is shown. It is considered that XY is specified when *rankmap* is not specified.

```
[two dimensions shape] --mpi "rank-map-bynode[={XY|YX}]"
[two dimensions shape] --mpi "rank-map-bychip[:{XY|YX}]"
```

Figure 2.24 The order of assigning node in two dimensions



The assigning node is done for the job that specifies the node shape by two dimensions in order shown in figure above.

When 2x2 is specified for the shape parameter of --mpi option and eight is specified for the proc parameter, each process assigning of the rank-map-bynode parameter and the rank-map-bychip parameter becomes it as follows.

Table 2.30 When you specify "--mpi shape=2x2 proc=8" by assigning two dimensions the node

The order of assigning node	Assigned rank	
	For rank-map-bynode	For rank-map-bychip
0th	rank 0 and 4	rank 0 and 1
1st	rank 1 and 5	rank 2 and 3
2nd	rank 2 and 6	rank 4 and 5
3rd	rank 3 and 7	rank 6 and 7

Figure 2.25 For rank-map-bynode (two dimensions)

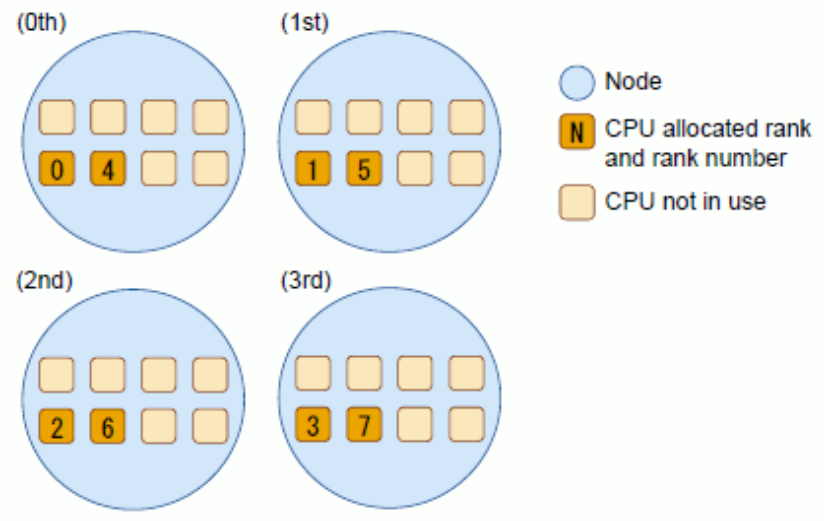
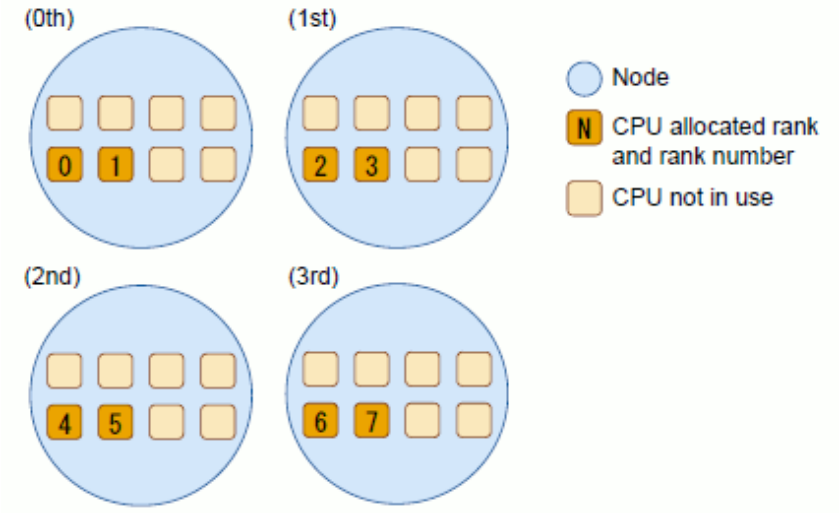


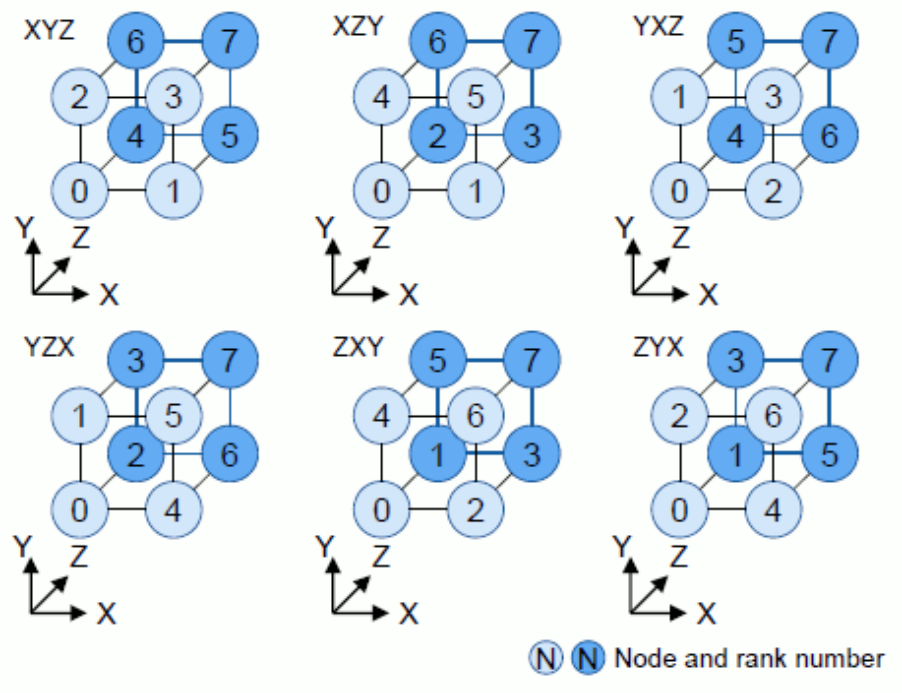
Figure 2.26 For rank-map-bychip (two dimensions)



Next, the image of the node assigned when three dimensions shape is specified is shown. It is considered that XYZ is specified when *rankmap* is not specified.

```
[three dimensions shape] --mpi "rank-map-bynode[={XYZ|XZY|YXZ|YZX|ZXY|ZYX}]"
[three dimensions shape] --mpi "rank-map-bychip[:{XYZ|XZY|YXZ|YZX|ZXY|ZYX}]"
```

Figure 2.27 The order of assigning node in three dimensions



The assigning node is done for the job that specifies the node shape by three dimensions in order shown in figure above.

When 2x2x2 is specified for the shape parameter of --mpi option and 16 is specified for the proc parameter, each process assigning of the rank-map-bynode parameter and the rank-map-bychip parameter becomes it as follows.

Table 2.31 When you specify "--mpi shape=2x2x2 proc=16" by assigning three dimensions the node

The order of assigning node	Assigned rank	
	For rank-map-bynode	For rank-map-bychip
0th	rank 0 and 8	rank 0 and 1
1st	rank 1 and 9	rank 2 and 3
2nd	rank 2 and 10	rank 4 and 5
3rd	rank 3 and 11	rank 6 and 7
4th	rank 4 and 12	rank 8 and 9
5th	rank 5 and 13	rank 10 and 11
6th	rank 6 and 14	rank 12 and 13
7th	rank 7 and 15	rank 14 and 15

Figure 2.28 For rank-map-bynode (three dimensions)

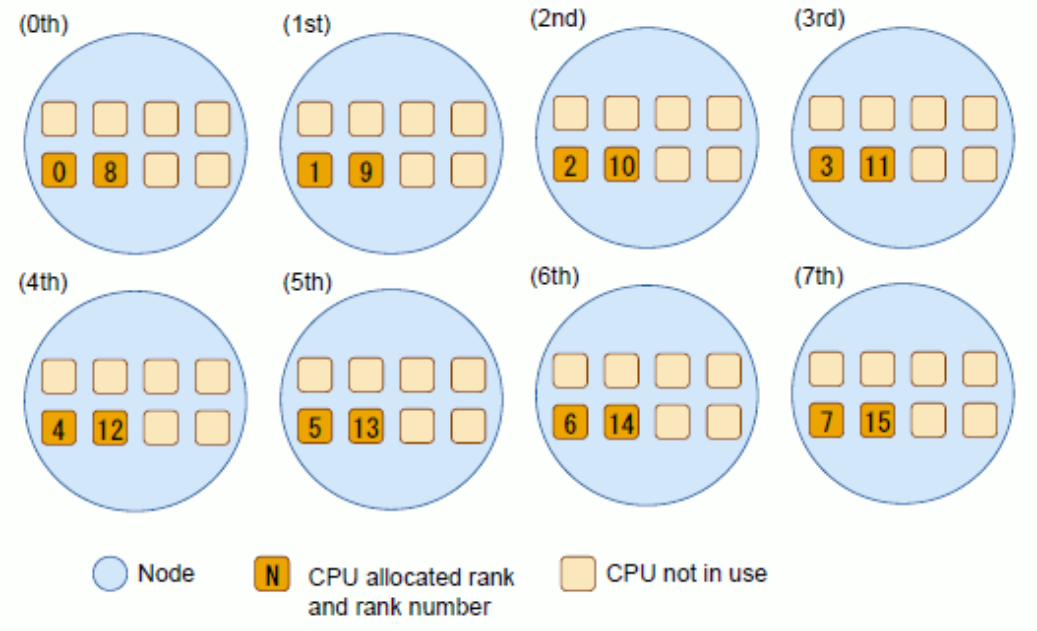
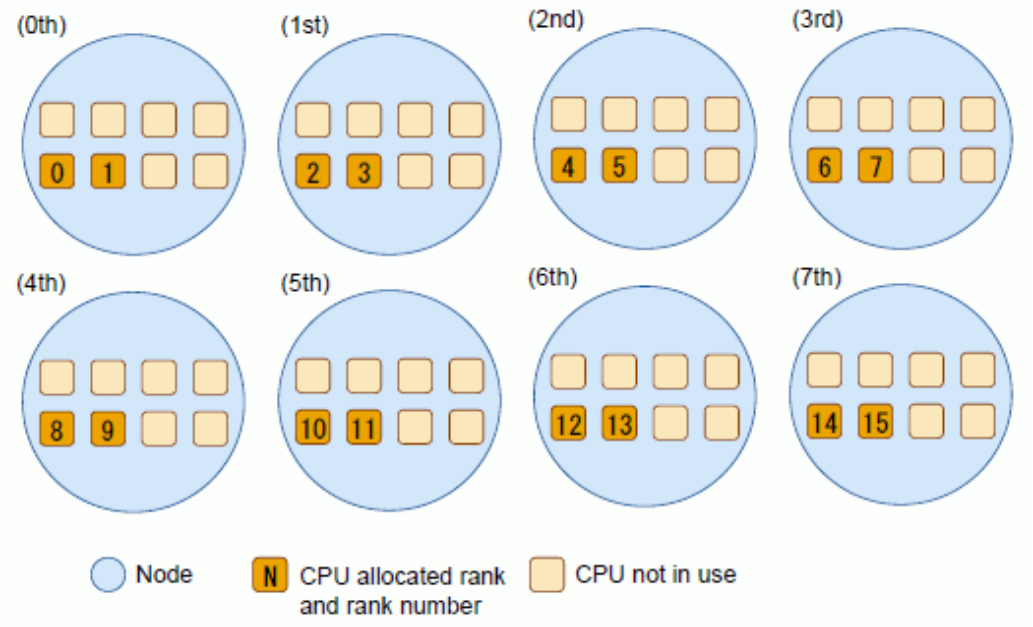


Figure 2.29 For rank-map-bychip (three dimensions)



2.3.6.7 Node specification by rank-map-hostfile parameter [FX]

If you want to specify the order of selection of nodes assigned to ranks, specify the rank-map-hostfile parameter.

The followings are the formats of rank-map-hostfile parameter.

```
--mpi "rank-map-hostfile=filename"
--mpi "rank-map-bychip,rank-map-hostfile=filename"
--mpi "rank-map-bynode,rank-map-hostfile=filename"
```

MPI assigns the node at coordinates written in the file *filename* to a rank.

The node is specified together with one-dimensional, two-dimensional, or three-dimensional coordinates, according to the process shape.

Write one set of coordinates in parentheses per line in a file *filename*.

Table 2.32 How to write a rank-map-hostfile parameters

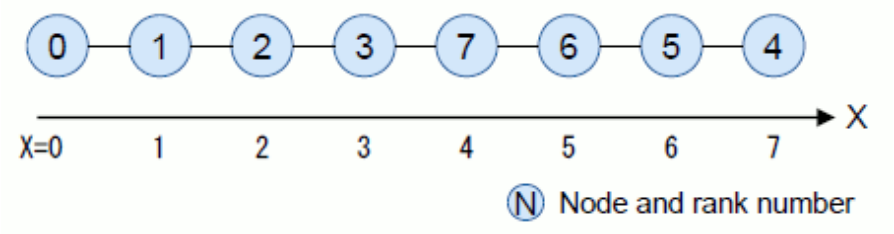
To specify a coordinate	Content describe
one-dimensional coordinate	(X)
two-dimensional coordinate	(X, Y)
three-dimensional coordinate	(X, Y, Z)

The reading permission to the user who submits the job is necessary for a file *filename*.

The following example shows specification of one-dimensional rank assignment using the rank-map-hostfile parameter.

```
$ cat rankmapfile-1
(0)
(1)
(2)
(3)
(7)
(6)
(5)
(4)
$ pjsub -L "node=8" --mpi "rank-map-hostfile=rankmapfile-1" job.sh
```

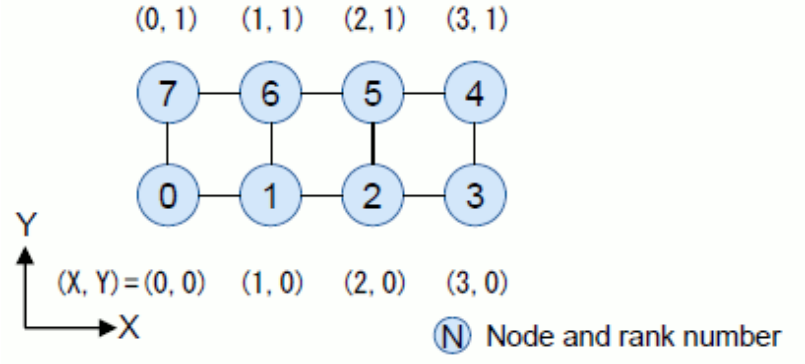
Figure 2.30 Assigning node figure of one dimension using rank-map-hostfile parameter



The following example shows specification of two-dimensional rank assignment using the rank-map-hostfile parameter.

```
$ cat rankmapfile-2
(0,0)
(1,0)
(2,0)
(3,0)
(3,1)
(2,1)
(1,1)
(0,1)
$ pjsub -L "node=4x2" --mpi "rank-map-hostfile=rankmapfile-2" job.sh
```

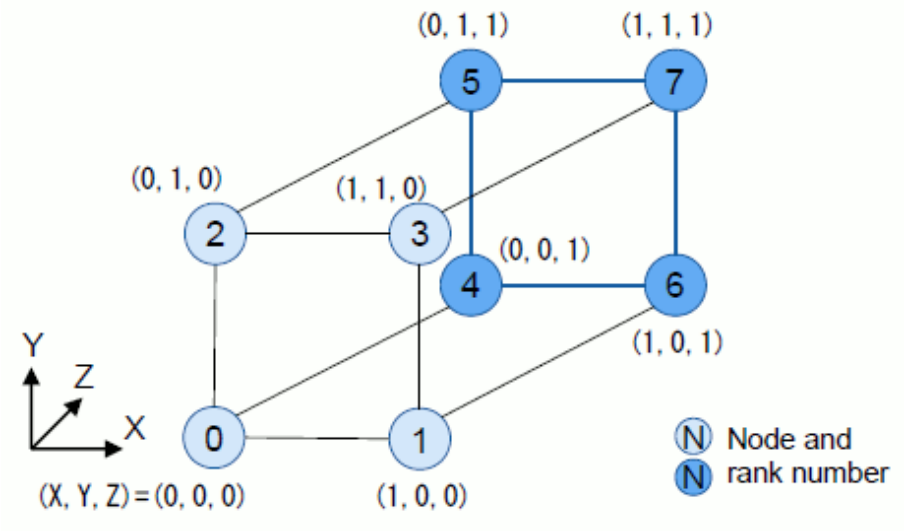
Figure 2.31 Assigning node figure of two dimensions using rank-map-hostfile parameter



The following example shows specification of three-dimensional rank assignment using the rank-map-hostfile parameter.

```
$ cat rankmapfile-3
(0,0,0)
(1,0,0)
(0,1,0)
(1,1,0)
(0,0,1)
(0,1,1)
(1,0,1)
(1,1,1)
$ pjsub -L "node=2x2x2" --mpi "rank-map-hostfile=rankmapfile-3" job.sh
```

Figure 2.32 Assigning node figure of three dimensions using rank-map-hostfile parameter



Note

- The blank line under the file *filename* is disregarded.
- The coordinates written in the file *filename* must be values within the range specified by the shape parameter representing the node shape. As an example, for shape=2x3, the coordinates (0,0), (0,1), (0,2), (1,0), (1,1), and (1,2) can be written.
- To specify the rank-map-hostfile parameter together with the rank-map-bychip parameter, the coordinates written in the file *filename* must conform to the following:
  - The number of coordinates in the file *filename* must be equal to the number of nodes in the specified shape represented by the shape parameter.
    As an example, for shape=3x2, the number of nodes is 6, so write six coordinates in the file *filename*.
    If the number of written coordinates is less than the number of nodes in the specified shape represented by the shape parameter, the pjsub command rejects the acceptance of the job. If the number of written coordinates is greater than the number of nodes in the specified shape represented by the shape parameter, the extra coordinates are disregarded.
  - Multiple identical coordinates cannot be written in the file *filename*. If identical coordinates are written, the pjsub command returns an error.
- To specify the rank-map-hostfile parameter together with the rank-map-bynode parameter, the coordinates written in the file *filename* must conform to the following:
  - Suppose that the number of coordinates written in the file *filename* is less than the number of processes specified by the proc parameter. In this case, node assignment proceeds up to the node at the last coordinates and then returns to the node at the first coordinates.
    If the number of coordinates is greater than the specified number of processes in the proc parameter, the extra coordinates are disregarded.

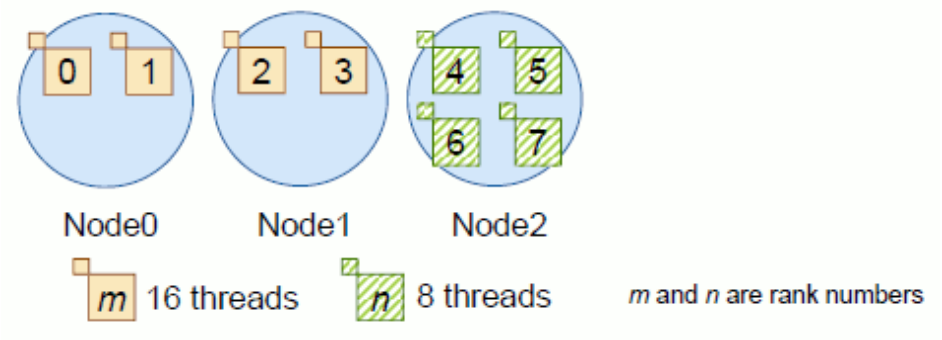
- The file *filename* may contain identical written coordinates if the number or coordinates is less than or equal to the number of CPU cores per compute node.

2.3.6.8 --vcoordfile option of the mpiexec command [FX]

The above-described rank-map-hostfile parameter is not the only method of specifying the allocation of nodes to ranks. Another method uses the --vcoordfile option of the mpiexec command.

The following example uses the --vcoordfile option of the mpiexec command to specify the coordinates of nodes and number of cores to allocate to each rank.

```
[Job Script]
#PJM -L node=3
#PJM --mpi max-proc-per-node=4
mpiexec --vcoordfile vfile a.out

[Contents of vfile file]
(0) core=16    <- Allocate 16 cores of node at coordinate (0) to rank 0
(0) core=16    <- Allocate 16 cores of node at coordinate (0) to rank 1
(1) core=16    <- Allocate 16 cores of node at coordinate (1) to rank 2
(1) core=16    <- Allocate 16 cores of node at coordinate (1) to rank 3
(2) core=8     <- Allocate 8 cores of node at coordinate (2) to rank 4
(2) core=8     <- Allocate 8 cores of node at coordinate (2) to rank 5
(2) core=8     <- Allocate 8 cores of node at coordinate (2) to rank 6
(2) core=8     <- Allocate 8 cores of node at coordinate (2) to rank 7
```



See

For details on the --vcoordfile option of the mpiexec command, see "MPI User's Guide," which is a Development Studio manual.

Information

The node allocated to each rank by the rank-map-hostfile parameter of the pjsub command must be different from all other nodes allocated by the command. In contrast, the --vcoordfile option of the mpiexec command can allocate the same node to multiple ranks when the CPU core has free space.

Also, as long as the rank-map-hostfile parameter is commonly specified for the MPI programs executed in a job, you can change the specification of the --vcoordfile option for each MPI program executed (in each execution of the mpiexec command).
The specified allocation of a node to a rank with the --vcoordfile option has priority over that with the rank-map-hostfile parameter.

2.3.6.9 Standard output/standard error output of the mpiexec command [FX]

The standard output and standard error output of a parallel process run with the mpiexec command can optionally specify the destination file. If the output destination is omitted, the job ACL function setting applies (See "Default behavior for standard output/standard error

output of the mpiexec command"). Note that the standard output and standard error output of the mpiexec command are the standard output and standard error output of the job, respectively.

Output destinations can be specified for each mpiexec command or for each parallel process.

- Output for each mpiexec command
  The outputs of each parallel process are output to the same specified file.
  - mpiexec { -of | --of | -std | --std } *stdfile*

    Outputs the standard output and standard error output of parallel processes to the file *stdfile*.
  - mpiexec { -ofout | --ofout | -stdout | --stdout } *out_file*

    Outputs the standard output of parallel processes to the file *out_file*.
  - mpiexec { -oferr | --oferr | -stderr | --stderr } *err_file*

    Outputs the standard error output of parallel processes to the file *err_file*.
- Output for each parallel process

  The outputs of each parallel process are output to a different file for each rank (process).
  The file name is "*SpecifiedName.mpiexec.rank*.", where *mpiexec* is the number of times the mpiexec command is executed in the job and *rank* is the rank number of the parallel process.
  - mpiexec { -of-proc | --of-proc | -std-proc | --std-proc } *proc_file*

    Outputs the standard output and standard error output of parallel processes for each rank to the file *proc_file.mpiexec.rank*.
  - mpiexec { -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc } *out_proc_file*

    Outputs the standard output of parallel processes for each rank to the file *out_proc_file.mpiexec.rank*.
  - mpiexec { -oferr-proc | --oferr-proc |-stderr-proc | --stderr-proc } *err_proc_file*

    Outputs the standard error output of parallel processes for each rank to the file *err_proc_file.mpiexec.rank*.

The following metacharacters can be used in the file name.

Table 2.33 Metacharacters that can be used in the destination filename for the mpiexec command (1)

Metacharacter	Description
%j	Job ID
%J	Subjob ID
%b	Bulk number For other than bulk job, this is replaced with empty.
%s	Step number For other than step job, this is replaced with empty.
%n	Job name
%o	The standard output file name for the job For an interactive job, the file name is "./%n.%J.out".
%e	The standard error output file name for the job For an interactive job, the file name is "./%n.%J.err".
%m	Number of times the mpiexec command runs in the job
%r	Rank number and spawn number For static process: rank number For dynamic process: rank number@spawn number If the destination is for each mpiexec command, this is replaced with empty.
%R	Rank number If the destination is for each mpiexec command, this is replaced with empty.
%S	spawn number For static process: 0

Metacharacter	Description
	For dynamic process: spawn number If the destination is for each mpiexec command, this is replaced with empty.

For %r, %R, and %S, the following formats can also be used.

Table 2.34 Metacharacters that can be used in the destination filename for the mpiexec command (2)

Format	Description	Example
%0*N*r %0*N*R %0*N*S	If the display string of the rank or spawn number is less than the minimum field width N, it is padded with 0.	[For rank 3] %02r : 03 (for static process) %02r : 03@01 (for dynamic process) %02R : 03 %02S : 01
%/*N*r %/*N*R %/*N*S	Rounds a rank or spawn number down to the nearest *N*.	[For rank 3] %/100r : 0 (for static process) %/100r : 0@0 (for dynamic process) %/100R : 0 %/100S : 0 [For rank 203] %/100r : 200 (for static process) %/100r : 200@0 (for dynamic process) %/100R : 200 %/100S : 0
%0*M*/Nr %0*M*/*N*R %0*M*/*N*S	Rounds a rank or spawn number down to the nearest *N*. If the display string of the rounded value is less than the minimum field width *M*, it is padded with 0.	[For rank 3] %04/100r : 0000 (for static process) %04/100r : 0000@0000 (for dynamic process) %04/100R : 0000 %04/100S : 0000 [For rank 203] %04/100r : 0200 (static process) %04/100r : 0200@0000 (for dynamic process) %04/100R : 0200 %04/100S : 0000

Here are some examples of specifications and output files. In this case, assume that the job ID is 123.

Example 1) Output standard output and standard error output to respective files

```
mpiexec --stdout ./%j.stdout --stderr ./%j.stderr ./a.out
```

```
$ ls
123.stderr 123.stdout a.out
```

Example 2) Output to files for each rank

```
mpiexec -stdout-proc ./%j.stdout -stderr-proc ./%j.stderr ./a.out
```

```
$ ls
123.stderr.1.0 123.stderr.1.1 123.stdout.1.0 123.stdout.1.1 a.out
```

Example 3) Change the output directory every 10 ranks

```
mpiexec -stdout-proc ./%/10R/%j.stdout -stderr-proc ./%/10R/%j.stderr ./a.out
```

```
$ ls
0/ 10/ a.out
$ ls 0/
123.stderr.1.0 123.stderr.1.1 ... 123.stderr.1.9
123.stdout.1.0 123.stdout.1.1 ... 123.stdout.1.9
```

Example 4) Output to files for each spawn number

```
mpiexec -stdout-proc ./%S/%j.stdout -stderr-proc ./%S/%j.stderr ./a.out
```

```
$ ls
0/ 1/ a.out
$ ls 0/
123.stderr.1.0 123.stderr.1.1 ... 123.stderr.1.9
123.stdout.1.0 123.stdout.1.1 ... 123.stdout.1.9
$ ls 1/
123.stderr.1.0@1 123.stderr.1.1@1 ... 123.stderr.1.9@1
123.stdout.1.0@1 123.stdout.1.1@1 ... 123.stdout.1.9@1
```

Default behavior for standard output/standard error output of the mpiexec command

The default behavior for the standard output/standard error output of the mpiexec command is set in the job ACL function. Use the pjacl command to check the setting.

Table 2.35 Default behavior for standard output/standard error output of the mpiexec command

Behavior	Iitem of job ACL function	Value
Output unit	mpiexec-stdouterr-unit	mpiexec : Output for each mpiexec command proc : Output for each parallel process (rank)
Standard output (Batch job)	mpiexec-stdout	*filename* : Output file. "*filename*" may contain metacharacters. noset : Standard output/Standard error output of the mpiexec command The job ACL feature may not allow change of the output. Check the parameter "mpiexec(xxxx)" of the item "execute" in the job ACL function.
Standard error output (Batch job)	mpiexec-stderr	
Standard output (Interactive job)	mpiexec-stdout(interact)	
Standard error output (Interactive job)	mpiexec-stderr(interact)	
Behavior if no output	mpiexec-std-emptyfile	on : Create empty file off : Do not create empty file To change this behavior, set the environment variable PLE_MPI_STD_EMPTYFILE. Example: PLE_MPI_STD_EMPTYFILE="off" For the value "force-on" or "force-off", the behavior is follows: force-on : Ignore the PLE_MPI_STD_TMPTYFILE and create empty file. force-off : Ignore the PLE_MPI_STD_TMPTYFILE and do not create empty file.

See

See also "2.2.2 Checking restriction information" for the output of the pjacl command.

Notes on running large-scale MPI jobs

For MPI jobs that are highly parallel (Generates approximately 10000 or more MPI processes) and output standard output or standard error output to a file for each rank, it is recommend to run them as follows, considering the system load of writing to a file:

- Output the standard output/standard error output of the mpiexec command to a different file for each rank (process).
- Output the standard output/standard error output files for each rank to a different directory for every several files, rather than to the same directory.
- Do not create an empty file if there is no standard output/standard error output for ranks.

Example: Change the output directory for standard output and standard error output every 1000 rank numbers. Also, an empty file is not created if there is no standard output or standard error output.

```
export PLE_MPI_STD_EMPTYFILE="off"
mpiexec -stdout-proc ./%/1000R/%j.stdout -stderr-proc ./%/1000R/%j.stderr ./a.out
```

2.3.6.10 Environment variable in MPI processes [FX]

The Job Operation Software sets the following environment variable in MPI processes.

Table 2.36 Environment variable in MPI processes

Environment variable	Description
PMIX_RANK	The rank number of the MPI process is set in decimal.
PLE_RANK_ON_NODE	The identification number of the MPI process in the compute node is set in decimal. An identification number is a unique number assigned within an MPI process belonging to the same MPI_COMM_WORLD on the same compute node, starting with 0.

2.3.7 Examples of specifying MPI job execution

This section describes examples of executing MPI jobs as node allocated jobs.

See

- For details on how to use the mpiexec command to execute an MPI program, see "MPI User's Guide," which is the Development Studio manual.
- For details on executing programs of an MPI processing system other than Development Studio, see "Appendix C Executing programs of MPI processing system other than Development Studio."

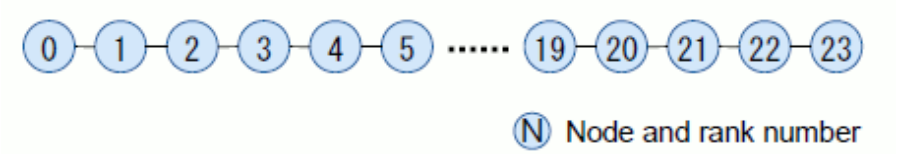
2.3.7.1 Executing a job in a one-dimensional node shape

The following example shows execution of the MPI program prog_A by the mpiexec command. 24 processes are continuously mapped in one-dimensional nodes.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=24"                                                    (1)
#PJM --mpi "shape=24"                                                (2)
...
mpiexec -n 24 ./prog_A                                               (3)
$ pjsub job.sh
```

(1) Node shape: one-dimensional with 24 nodes
(2) Process shape: one-dimensional with 24 nodes
(3) Executes prog_A with 24 parallel processes.

Figure 2.33 Executing a job in a one-dimensional shape



With the following specifications, 24 processes can also be mapped in one-dimensional nodes.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=24"                                                    (1)
...
mpiexec -n 24 ./prog_A                                               (2)
$ pjsub job.sh
```

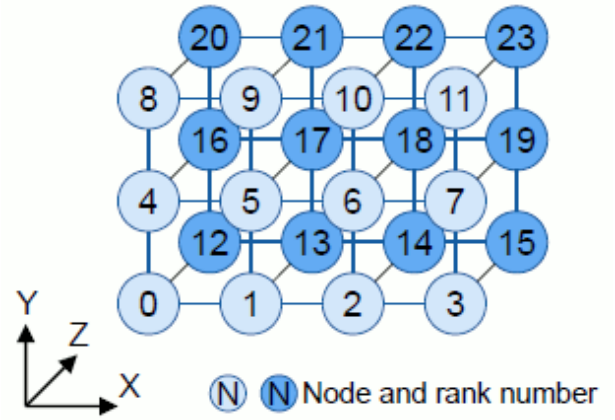
(1) Node shape: one-dimensional with 24 nodes
(2) Executes prog_A with 24 parallel processes. The process shape is the same as the node shape with -L node=24.

2.3.7.2 Executing a job in a three-dimensional node shape

The following example shows execution of the MPI program prog_A by the mpiexec command. 24 processes are continuously mapped in three-dimensional nodes.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=4x3x2"                  <- Node shape: three-dimensional with 24 nodes
...
mpiexec ./prog_A
$ pjsub job.sh
```

Figure 2.34 Executing a job in a three-dimensional shape



2.3.7.3 Executing a program several times in one job

When using a single job to execute an MPI program several times, be careful with the required number of nodes.

For the node shape specified in the pjsub command (the specified value of --mpi shape parameter, or the specified value of -L node if shape parameter is omitted), specify the maximum number of processes.

Using the mpiexec command option (-n, --n, -np or --np) for specifying the number of processes, specify an arbitrary value that is less than the maximum number of processes.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=4x3x2"                                                 (1)
...
mpiexec -n 12 ./prog_A                                               (2)
```

```
mpiexec ./prog_B                                                    (3)
mpiexec -n 16 ./prog_C                                              (4)
$ pjsub job.sh
```

(1) Three-dimensional with 24 nodes
(2) 12 parallel processes
(3) Number of parallel processes: Maximum number of created processes (24)
(4) 16 parallel processes

First, determine the rank assignment, and use it until the specified process.

Figure 2.35 Determining the rank assignment

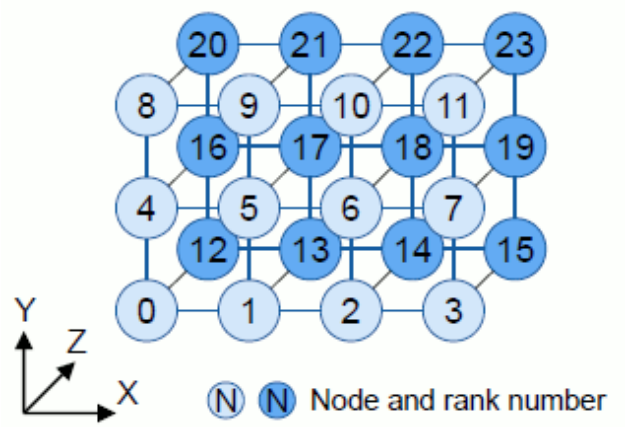
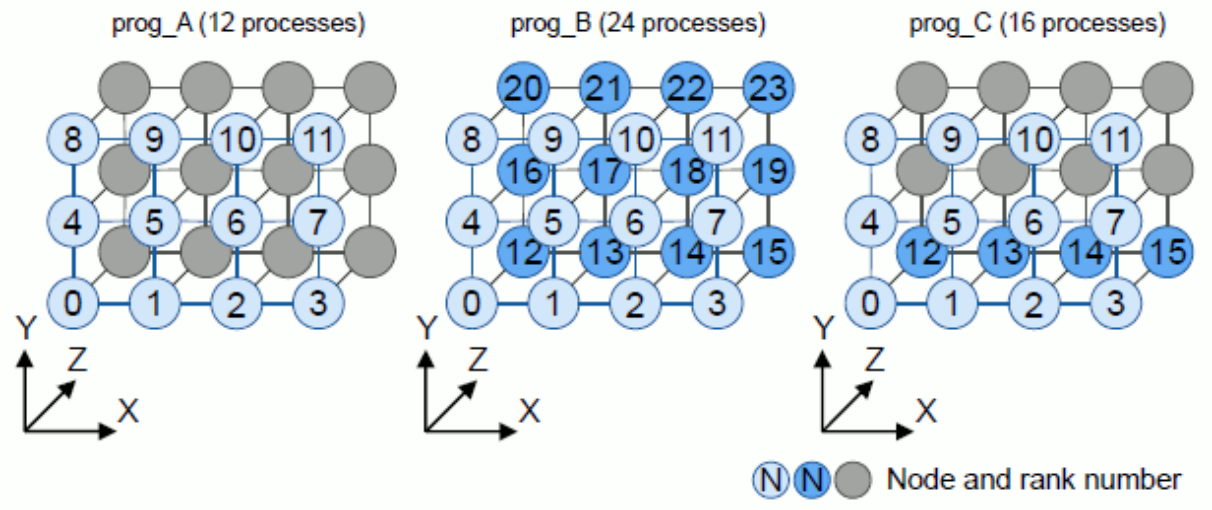


Figure 2.36 Process execution example



2.3.7.4 Example of executing multi-processes in one node job that specifies rank-map-bynode parameter and rank-map-hostfile parameter

The example with the rank-map-bynode and the rank-map-hostfile parameters, and the job creates 2 processes on one node is executed is shown below.

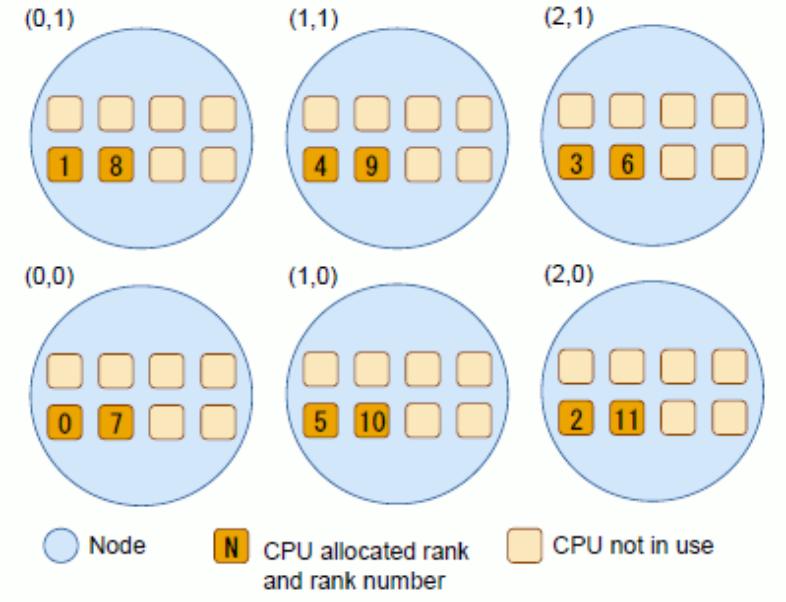
In the following examples, the node shown by coordinates of the first line of *hostfile* is assumed to be rank 0, and the node is assigned every one line.

```
$ cat hostfile
(0, 0)
(0, 1)
(2, 0)
(2, 1)
(1, 1)
```

```
(1, 0)
(2, 1)
(0, 0)
(0, 1)
(1, 1)
(1, 0)
(2, 0)
$ cat job.sh
#!/bin/sh
#PJM -L "node=3x2"                  <- Two-dimensional with 6 nodes.
#PJM --mpi "rank-map-hostfile=hostfile"
#PJM --mpi "proc=12"
#PJM --mpi "rank-map-bynode"
...
mpiexec ./prog_A
```

The number of processes created to each one node becomes 12/ (3x2) =2.

Figure 2.37 Assigning node order of using a rank-map-bynode parameter and rank-map-hostfile parameter



2.3.7.5 Example of executing multi-processes in one node job that specifies rank-map-bychip parameter and rank-map-hostfile parameter

The example with the rank-map-bychip and the rank-map-hostfile parameters, and the job creates 2 processes on one node is executed is shown below.

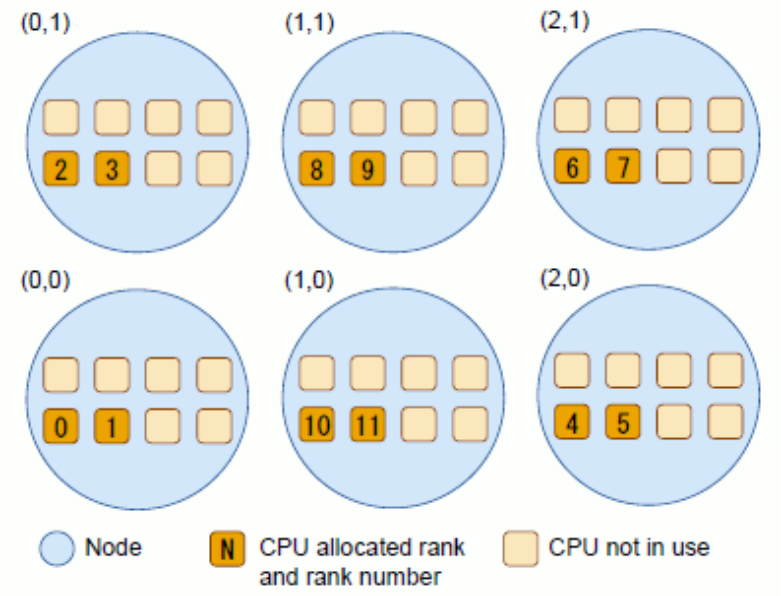
In the following examples, the node shown by coordinates of the first line of the *hostfile* file is assumed to be rank 0, and the number specified by the rank-map-bychip parameter is assigned. The remaining line in the *hostfile* file is disregarded.

```
$ cat hostfile
(0, 0)
(0, 1)
(2, 0)
(2, 1)
(1, 1)
(1, 0)
(2, 1)
(0, 0)
(0, 1)
(1, 1)
(1, 0)
(2, 0)
```

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=3x2"                  <- Two-dimensional with 6 nodes.
#PJM --mpi "rank-map-hostfile=hostfile"
#PJM --mpi "proc=12"
#PJM --mpi "rank-map-bychip=2"
...
mpiexec ./prog_A
$ pjsub job.sh
```

The number of processes created to one node becomes 12/ (3x2) =2.

Figure 2.38 Assigning node order of using a rank-map-bychip parameter and rank-map-hostfile parameter



2.3.7.6 How to execute an MPI program in the MPMD model

The following example shows specification of execution of an MPI program consisting of several different programs. It is an MPI program in the MPMD (Multiple Program Multiple Data) model.

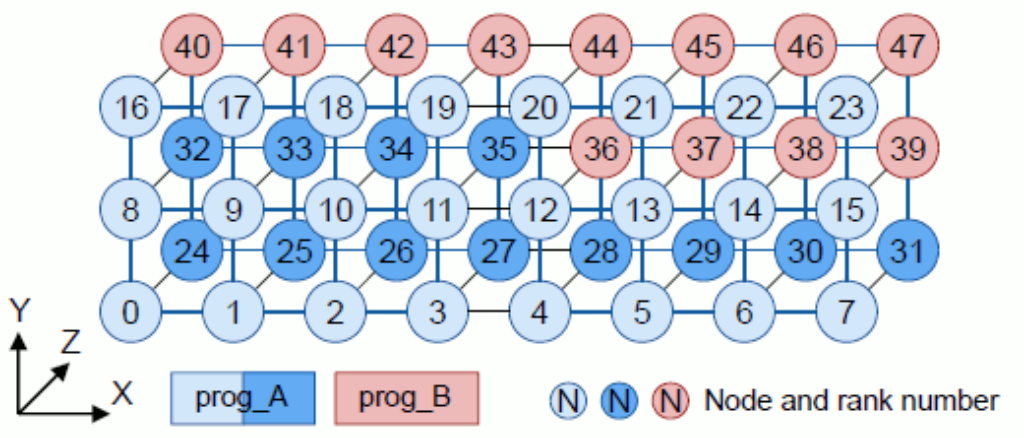
For the number of processes specified in the pjsub command (the specified value of --mpi shape, or the specified value of -L node if shape is omitted), specify the total parallelization value (number of processes) of each program. A combination of the number of processes to be generated and the MPI program name are delimited by a colon.

Using the option (-n) for specifying the number of mpiexec processes, specify the parallelization of each program (number of processes). If the -n option is not specified, the value specified in the -L node option is passed. Consequently, the specified number would exceed the total number of nodes, and the job would end in an error.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=8x3x2"                                                    (1)
...
mpiexec -n 36 ./prog_A : -n 12 ./prog_B                                 (2)
$ pjsub job.sh
```

(1) Three-dimensional with 48 nodes. Total number of nodes required for prog_A and prog_B
(2) The option for specifying parallel processes cannot be omitted for MPMD.

Figure 2.39 Specifying sequential execution of a job



2.3.7.7 Specifying a rank for an MPI program in the MPMD model

You can specify rank assignment for an MPI program in the MPMD model by using the --mpi rank-map-hostfile parameter of the pjsub command.

Using a colon as a delimiter, specify a combination of the number of processes to be generated and the MPI program name in the mpiexec command.

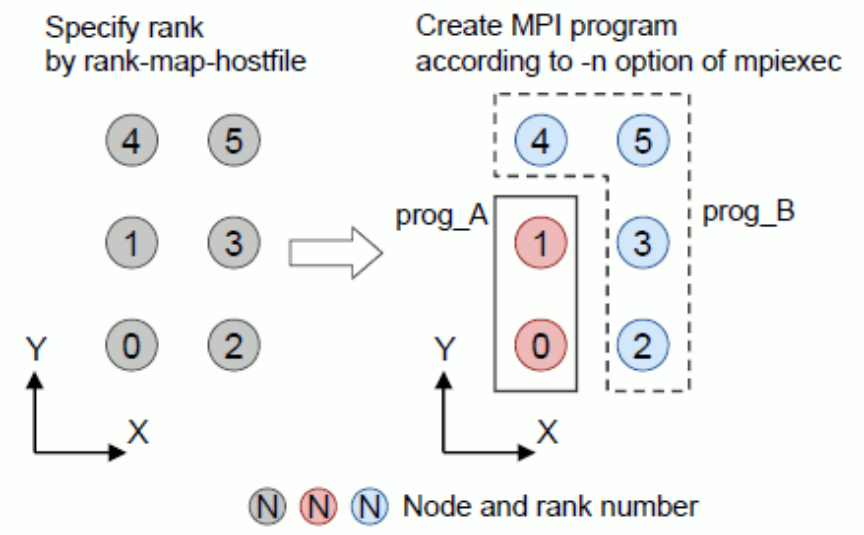
The job will assign a process from the beginning of the *filename* file.

The following is a specification example.

```
$ cat filename                                                    (1)
(0,0)
(0,1)
(1,0)
(1,1)
(0,2)
(1,2)
$ cat job.sh
#!/bin/sh
#PJM -L "node=2x3"
#PJM --mpi "rank-map-hostfile=filename"                           (2)
...
mpiexec -n 2 ./prog_A : -n 4 ./prog_B                             (3)
$ pjsub job.sh
```

(1) File specifying the rank map
(2) Specifies the rank map.
(3) Executes the MPI programs prog_A and prog_B in the MPMD model.

Figure 2.40 Assigning an MPMD program shape



2.3.7.8 Executing multiple MPI programs on the same node (static processes) [FX]

This section describes an example of executing multiple MPI programs in one job. To execute multiple MPI programs, execute the mpiexec command in the background.

```
#!/bin/sh
#PJM -L node=4
#PJM --mpi max-proc-per-node=8
mpiexec --vcoordfile file_a a.out &  <- MPI program a.out
mpiexec --vcoordfile file_b b.out &  <- MPI program b.out
mpiexec --vcoordfile file_c c.out    <- MPI program c.out
```

In this example, the nodes specified in the vcoordfile option of the mpiexec command respectively execute the MPI programs a.out, b.out, and c.out.

The number of processes in the MPI program a.out is four. Allocate four cores to each rank of the program.

```
[File file_a]
(0) core=4    <- Allocate 4 cores of node at coordinate (0) to rank 0 of a.out
(1) core=4    <- Allocate 4 cores of node at coordinate (1) to rank 1 of a.out
(2) core=4    <- Allocate 4 cores of node at coordinate (2) to rank 2 of a.out
(3) core=4    <- Allocate 4 cores of node at coordinate (3) to rank 3 of a.out
```

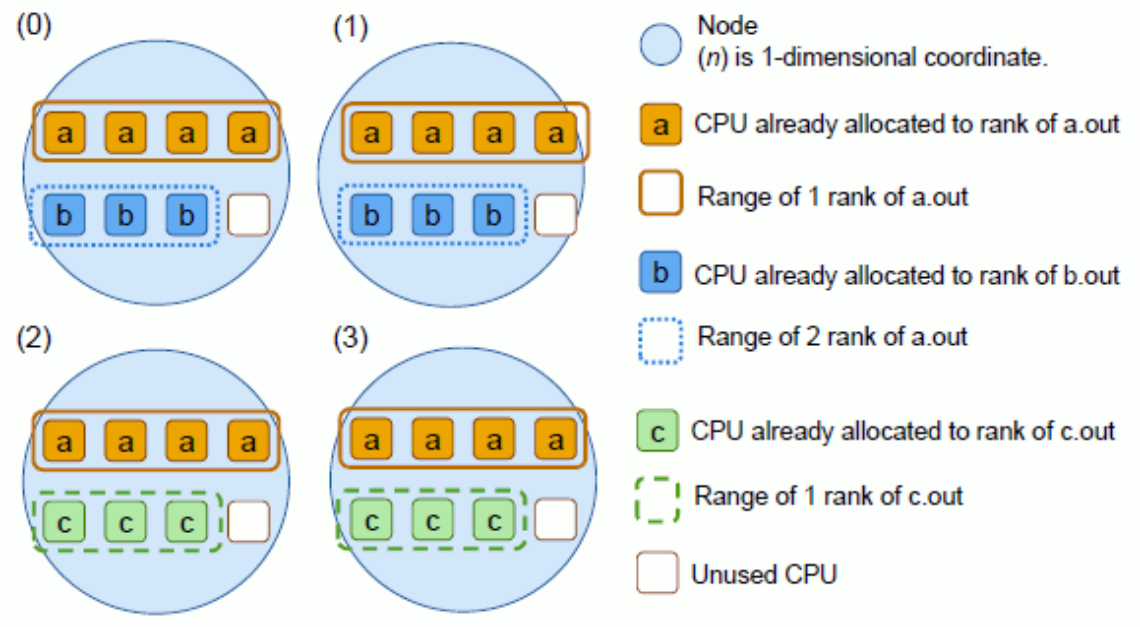
The number of processes in the MPI program b.out is two. Allocate three cores to each rank of the program.

```
[File  file_b]
(0) core=3     <- Allocate 3 cores of node at coordinate (0) to rank 0 of b.out
(1) core=3     <- Allocate 3 cores of node at coordinate (1) to rank 1 of b.out
```

The number of processes in the MPI program c.out is two. Allocate three cores to each rank of the program.

```
[File file_c]
(2) core=3     <- Allocate 3 cores of node at coordinate (2) to rank 0 of c.out
(3) core=3     <- Allocate 3 cores of node at coordinate (3) to rank 1 of c.out
```

Figure 2.41 Rank allocation when executing multiple MPI programs on the same node



2.3.7.9 Executing multiple MPI programs on the same node (dynamic processes) [FX]

This section introduces two examples of generating dynamic processes on nodes that already have existing MPI processes.

The following example shows execution in a job to generate a dynamic process on a node that has an existing static process.

```
[Job Script]
#PJM -L node=2
#PJM --mpi max-proc-per-node=8
mpiexec --vcoordfile vfile_a a.out     <- Generate static process
```

To generate a dynamic process, this example uses info key:vcoordfile of the MPI_Comm_spawn function to specify the rank placement of the destination node for the generated dynamic process.

The number of processes in the MPI program a.out is two. Allocate four cores to each rank of the program. Specify the node and number of cores to allocate to each rank in the vfile_a file.

```
[File  vfile_a]
(0) core=4    <- Allocate 4 cores of node at coordinate (0) to rank 0 of a.out
(1) core=4    <- Allocate 4 cores of node at coordinate (1) to rank 1 of a.out
```

The following example shows the the source code (a.c program written in C) of the MPI program a.out.

This example uses the MPI_Comm_spawn function to generate the dynamic process b.out. The number of processes in the dynamic process b.out is two. Allocate two cores to each rank of the process. Then, the b.out process is generated on the node that has an existing process of the MPI program a.out. Use the FJMPI_Topology_get_coords function so that the process obtains the node coordinates by itself.

See

For details on the FJMPI_Topology_get_coords function, see "MPI User's Guide," which is a Development Studio manual.

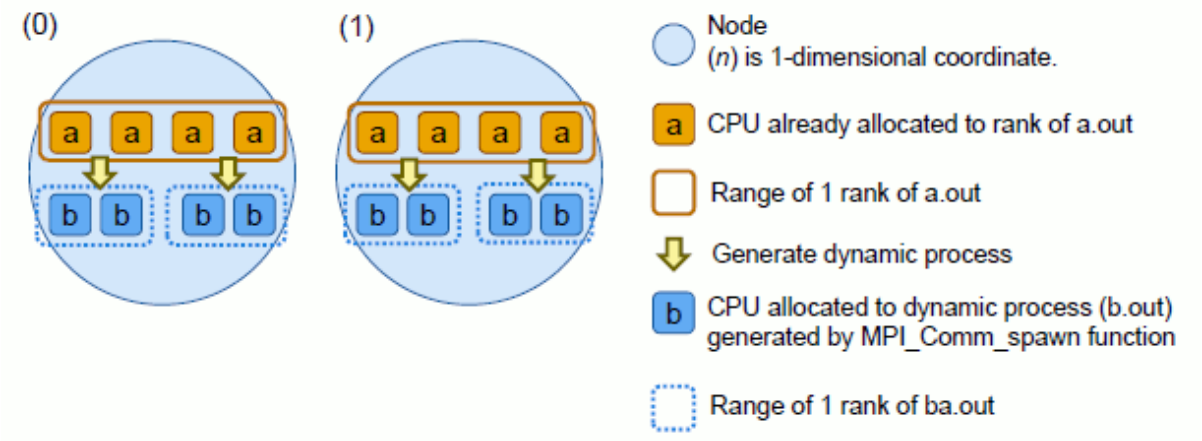
```
[a.c]
#include <mpi.h>
#include <mpi-ext.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    MPI_Info info;
```

```
    MPI_Comm comm;
    int rank;
    FILE *fp = NULL;
    char vfile[256];
    int coord[1];

    MPI_Init(&argc, &argv);
    ...
    // Obtain rank number
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Obtain 1-dimensional coordinate of node where own process exists
    FJMPI_Topology_get_coords(MPI_COMM_WORLD, rank, FJMPI_LOGICAL, 1, coord);

    // Generate vcoordfile file
    sprintf(vfile, "vcoordfile_%d", rank);
    fp = fopen(vfile, "w");
    ...
    // Specify 1-dimensional coordinate of destination node for generated dynamic process
    fprintf(fp, "(%d) core=2\n", coord[0]);
    fprintf(fp, "(%d) core=2\n", coord[0]);
    fclose(fp);

    // Generate info key
    MPI_Info_create(&info);
    // Use vcoordfile to specify destination node for generated dynamic process
    MPI_Info_set(info, "vcoordfile", vfile);
    // Generate dynamic process (b.out is MPI program with 2 processes)
    MPI_Comm_spawn("./b.out", MPI_ARGV_NULL, 2, info, 0,
    MPI_COMM_SELF, &comm, MPI_ERRCODES_IGNORE);
    ...
    MPI_Finalize();
    return 0;
}
```

The node and number of cores to allocate to each rank are set in the vcoordfile_N file (N = 0 or 1), which is created before the MPI_Comm_spawn function is called to generate the MPI process b.out.

```
[File vcoordfile_N (N=0 or 1)]
(N) core=2    <- Allocate 2 cores of node at coordinate (0) to rank N of b.out
(N) core=2    <- Allocate 2 cores of node at coordinate (1) to rank N of b.out
```

Figure 2.42 Rank allocation when static and dynamic processes coexist in a node



The following example generates, in a job, another dynamic process on a node that has an existing dynamic process.

```
[Job Script]
#PJM -L node=4
#PJM --mpi max-proc-per-node=8
mpiexec --vcoordfile vfile_a a.out    <- Generate static process
```

To generate a dynamic process from another dynamic process, this example uses info key:vcoordfile of the MPI_Comm_spawn function to specify the rank placement of the destination node for the generated dynamic process. The number of processes in the MPI program a.out is four. Allocate eight cores to each rank of the program. Specify the node and number of cores to allocate to each rank in the file_a file.

```
[File vfile_a]
(0) core=8    <- Allocate 8 cores of node at coordinate (0) to rank 0 of a.out
(1) core=8    <- Allocate 8 cores of node at coordinate (1) to rank 1 of a.out
```

The following example shows the source code (a.c program written in C) of the MPI program a.out. This example uses the MPI_Comm_spawn function to generate the dynamic process b.out. The number of processes in the dynamic process b.out is two. Allocate two cores to one process.

```
[a.c]
#include <mpi.h>
#include <mpi-ext.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    MPI_Info info;
    MPI_Comm comm;
    int rank;
    FILE *fp = NULL;
    char vfile[256];
    int coord[1];

    MPI_Init(&argc, &argv);
    ...
    // Obtain rank number
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // Obtain 1-dimensional coordinate of node where own process exists
    FJMPI_Topology_get_coords(MPI_COMM_WORLD, rank, FJMPI_LOGICAL, 1, coord);

    // Generate vcoordfile file
    sprintf(vfile, "vcoordfile_b_%d", rank);
    fp = fopen(vfile, "w");
    ...
    // Specify 1-dimensional coordinate of destination node for generated
    // dynamic process a.out specifies coordinate (2) or (3)
    // to use 1-dimensional coordinate (0) and (1) exclusively
    fprintf(fp, "(%d) core=2\n", coord[0]+2);
    fprintf(fp, "(%d) core=2\n", coord[0]+2);
    fclose(fp);

    // Generate info key
    MPI_Info_create(&info);
    // Use vcoordfile to specify destination node for generated dynamic process
    MPI_Info_set(info, "vcoordfile", vfile);
    // Generate dynamic process (b.out is MPI program with 2 processes)
    MPI_Comm_spawn("./b.out", MPI_ARGV_NULL, 2, info, 0,
    MPI_COMM_SELF, &comm, MPI_ERRCODES_IGNORE);
    ...
    MPI_Finalize();
    return 0;
}
```

The node and number of cores to allocate to each rank are set in the vcoordfile_b_*N* file (*N*: 1-dimensional coordinate of the node), which is created before the MPI_Comm_spawn function is called to generate the dynamic process b.out.

```
[File vcoordfile_b_N (N is 1-dimensional coordinate of the node)]
(N) core=2    <- Allocate 2 cores of node at coordinate (N) to rank 0 of b.out
(N) core=2    <- Allocate 2 cores of node at coordinate (N) to rank 1 of b.out
```

The following example shows the source code (b.c program written in C) of the dynamic process b.out.

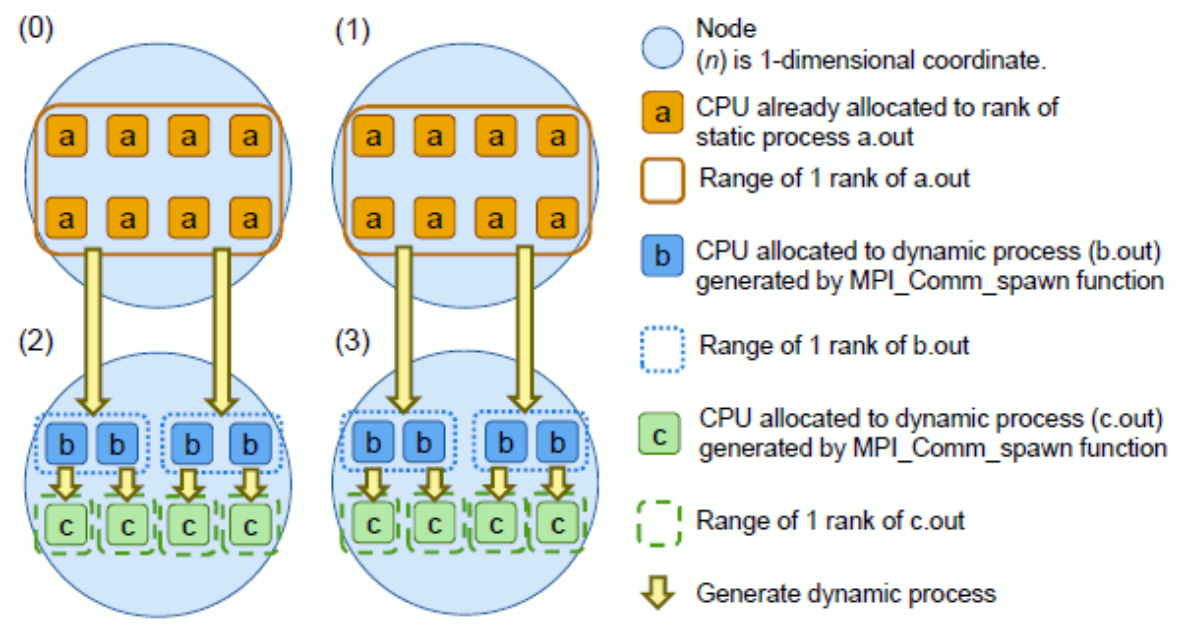
This example uses the MPI_Comm_spawn function to generate the dynamic process c.out. The number of processes in the dynamic process c.out is two. Allocate one core to one process.

```
[b.c]
#include <mpi.h>
#include <mpi-ext.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    MPI_Info info;
    MPI_Comm comm;
    int rank;
    FILE *fp = NULL;
    char vfile[256];
    int coord[1];

    MPI_Init(&argc, &argv);
    ...
    // Obtain rank number
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // Obtain 1-dimensional coordinate of node where own process exists
    FJMPI_Topology_get_coords(MPI_COMM_WORLD, rank, FJMPI_LOGICAL, 1, coord);

    // Generate vcoordfile file
    sprintf(vfile, "vcoordfile_c_%d", rank);
    fp = fopen(vfile, "w");
    ...
    // Specify 1-dimensional coordinate of destination node for generated dynamic process
    fprintf(fp, "(%d) core=1\n", coord[0]);
    fprintf(fp, "(%d) core=1\n", coord[0]);
    fclose(fp);

    // Generate info key
    MPI_Info_create(&info);
    // Use vcoordfile to specify destination node for generated dynamic process
    MPI_Info_set(info, "vcoordfile", vfile);
    // Generate dynamic process (c.out is MPI program with 2 processes)
    MPI_Comm_spawn("./c.out", MPI_ARGV_NULL, 2, info, 0,
    MPI_COMM_SELF, &comm, MPI_ERRCODES_IGNORE);
    ...
    MPI_Finalize();
    return 0;
}
```

The node and number of cores to allocate to each rank are set in the vcoordfile_c_*N* file (*N*: 1-dimensional coordinate of the node), which is created before the MPI_Comm_spawn function is called to generate the dynamic process c.out.

```
[File vcoordfile_c_N]
(N) core=1    <- Allocate 1 core of node at coordinate (N) to rank 0 of c.out
(N) core=1    <- Allocate 1 core of node at coordinate (N) to rank 1 of c.out
```

Figure 2.43 Rank allocation when multiple dynamic processes coexist in a node



2.3.7.10 Remote execution of a program [FX]

With the pjrsh command, you can execute a program at any compute node in the same job from an MPI process. Specify coordinates or an IP address on the shape of allocated compute nodes as the destination for the generated program.

- Specifying coordinates

```
pjrsh "coordinate" program arguments
```

The format of "*coordinate*" is as follows:

- One-dimensional coordinate: "(*x*)"
- Two-dimensional coordinates: "(*x,y*)"
- Three-dimensional coordinates: "(*x,y,z*)"

- Specifying an IP address

```
pjrsh IPaddress program arguments
```

Specify the IP address of a node in the *XXX.XXX.XXX.XXX* format in *IPaddress* to execute a program on the node. With the pjshowip command, you can obtain the IP addresses of the compute nodes allocated to a job. If the node of the specified IP address is not a compute node allocated to a job, the pjrsh command returns an error.

This section introduces two examples of remote execution. One example specifies a compute node with coordinates. The other example specifies it with an IP address.
The first example shows execution of a program from a process of the MPI program a.out with compute node coordinates specified.

```
[Job Script]
#PJM -L node=3
#PJM --mpi max-proc-per-node=4
mpiexec --vcoordfile vfile_a a.out    <- Generate static process
```

In this example, the vfile_a file specifies the coordinate of the node to allocate to each rank for the processes (number of processes: 8) generated at startup of the MPI program a.out.

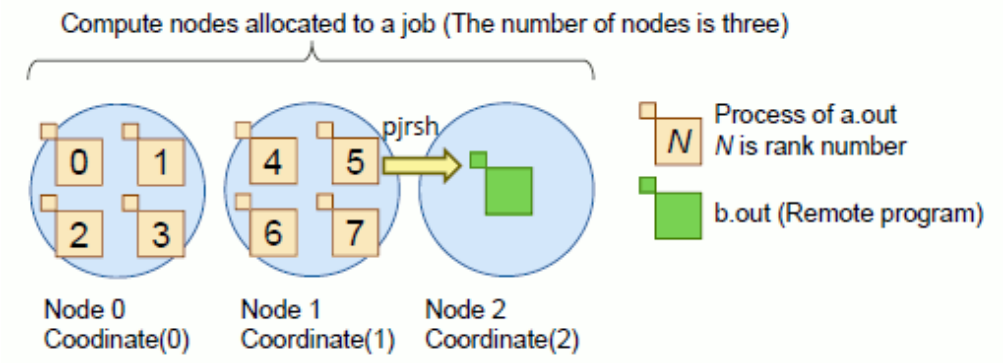
```
[File vfile_a]
(0)      <- Allocate a node at coordinate (0) to rank 0 of a.out
(0)      <- Allocate a node at coordinate (0) to rank 1 of a.out
(0)      <- Allocate a node at coordinate (0) to rank 2 of a.out
```

```
(0)      <- Allocate a node at coordinate (0) to rank 3 of a.out
(1)      <- Allocate a node at coordinate (1) to rank 4 of a.out
(1)      <- Allocate a node at coordinate (1) to rank 5 of a.out
(1)      <- Allocate a node at coordinate (1) to rank 6 of a.out
(1)      <- Allocate a node at coordinate (1) to rank 7 of a.out
```

To generate the process b.out (except MPI program) on the node at coordinate (2) from the process of rank 5 of the MPI program a.out, execute the following command using fork(2) and execve(2) in the MPI program a.out.

```
pjrsh "(2)" ./b.out
```

Figure 2.44 Illustration of remote execution of a program (1)



Note

- The envp argument of execve (2) must be specified all the calling environment variables.
  If you do not specify all environment variables, your program might not run correctly. For example, if you specify NULL for the envp argument of execve (2), the following error message is printed and the process generation fails:

  ```
  [ERR.] PLE 0002 plexec PLE service error occurred.(nid=own node)(CODE=code1, code2, code3)
  ```

- If b.out is an MPI program, use the MPI_Comm_spawn and MPI_Comm_spawn_multiple functions to execute b.out.

The next example shows execution of a program from a process of the MPI program a.out with a compute node IP address specified.

```
[Job Script]
#PJM -L node=2
#PJM --mpi max-proc-per-node=4
pjshowip > /foo/bar/ipaddr.lst      <- File that can also be referenced from MPI process
mpiexec --vcoordfile vfile_a a.out  <- Generate static process
```

In this example, the pjshowip command is executed in the job script to obtain a list of the IP addresses of allocated compute nodes and to save it in a file. Also, the vfile_a file specifies the coordinate of the node to allocate to each rank for the processes (number of processes: 6) generated at startup of the MPI program a.out.

```
[File vfile_a]
(0)      <- Allocate a node at coordinate (0) to rank 0 of a.out
(0)      <- Allocate a node at coordinate (0) to rank 1 of a.out
(0)      <- Allocate a node at coordinate (0) to rank 2 of a.out
(0)      <- Allocate a node at coordinate (0) to rank 3 of a.out
(1)      <- Allocate a node at coordinate (1) to rank 4 of a.out
(1)      <- Allocate a node at coordinate (1) to rank 5 of a.out
```

The list of the compute node IP addresses is output in the pjshowip command execution results. The output compute nodes are shown in order by rank number. Rank numbers are given according to the specification of the -L node=2 and --mpi max-proc-per-node=4 options of the pjsub command.

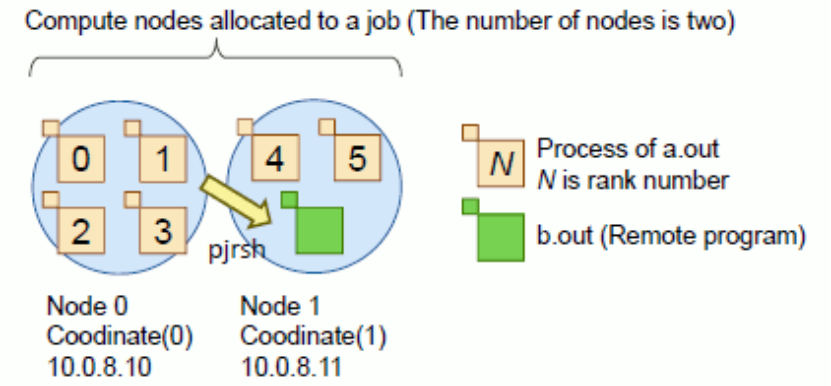
```
[File ipaddr.lst]
10.0.8.10   <- IP adress of a rank 0 compute node
10.0.8.10   <- IP adress of a rank 1 compute node
10.0.8.10   <- IP adress of a rank 2 compute node
10.0.8.10   <- IP adress of a rank 3 compute node
10.0.8.11   <- IP adress of a rank 4 compute node
10.0.8.11   <- IP adress of a rank 5 compute node
10.0.8.11   <- IP adress of a rank 6 compute node (*)
10.0.8.11   <- IP adress of a rank 7 compute node (*)
```

(*) In this example, the processes of ranks 6 and 7 are actually not generated because the number of processes in the MPI program a.out is six.

From the process of rank 1 of the MPI program a.out, to generate the process b.out on the node that has an existing process of rank 4, obtain the IP address of the node where rank 4 is placed. Obtain it from the /foo/bar/ipaddr.lst file, which saves the output results from the pjshowip command. Then, execute the following command using fork(2) and exec(2).

```
pjrsh 10.0.8.11 ./b.out
```

Figure 2.45 Illustration of remote execution of a program (2)



2.3.7.11 How to execute a hybrid parallel program

The following example shows execution of a hybrid parallel program that uses both process parallelism and thread parallelism.

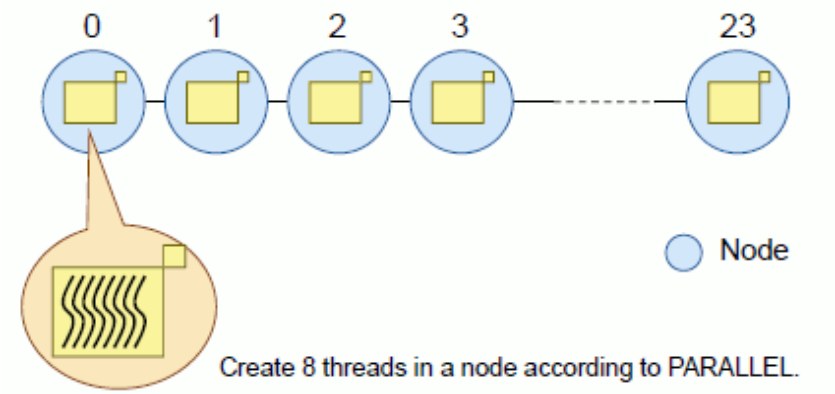
If the -n option of the mpiexec command is omitted, the value specified in the -L option of the pjsub command is automatically passed.

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=24"

export PARALLEL=8                   <- Specifies the number of threads.
...
mpiexec ./prog_A                    <- 24 parallel processes
$ pjsub job.sh
```

Note

In the above example, the environment variable PARALLEL specifies the number of threads in automatic parallelization. For details, see the Development Studio manual.

Figure 2.46 Hybrid parallel program generated



2.3.8 Specifying a job execution environment

To execute an application, specific software may need to be installed, as an example. Furthermore, another user may want to replicate the execution of an application, or you may still want to execute a conventional application in the old environment after a system upgrade. There may be other situations like that.

In addition to the default job execution environment (Normal mode), the Job Operation Software provides an environment in which job execution environments are explicitly specified and submitted so that applications using virtualization technologies such as Linux containers can run. End-users can specify an execution environment prepared in advance by the administrator to execute applications as jobs in this environment (SDI specification). End-users can also customize and build the execution environments by themselves (UDI specification). About the UDI specification, see the description of each job execution environment.

The Job Operation Software treats the job execution environment as a type of custom resource. Therefore, if there is a job execution environment you want to use, specify the corresponding custom resource when submitting a job. "jobenv" is the custom resource name of a job execution environment.

If the administrator has prepared multiple job execution environments, you can select one appropriate to your purpose. In this case, specify the name of the job execution environment as a value of the custom resource jobenv.

```
$ pjsub -L jobenv=container1 job1.sh   # Execute job1.sh in job execution environment container1
$ pjsub -L jobenv=container2 job2.sh   # Execute job2.sh in job execution environment container2
```

Use the pjacl command to check the job ACL function settings for the name of a job execution environment available to end-users. If the name of the job execution environment is omitted (-L jobenv), also check the job ACL function settings for the applied default job execution environment.

```
$ pjacl
#
# JOBACL information
#
...
pjsub option parameters
    ...
    (-L/--rsc-list)                    select     default
        ...
        (jobenv=)                      container1,container2    container1
        ...
```

In the above example, the available job execution environments are container1 and container2. If no job execution environment name is specified (pjsub -L jobenv), container1 is used. Contact your administrator to know which mode your system provides for the job execution environment.

Note

In the job execution environment, its startup process is run before the job execution. If the startup process takes a long time in the interactive job, the allocation of resources may time out as follows.

```
$ pjsub --interact -L "jobenv=container"
[INFO] PJM 0000 pjsub Job 497 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 497 started.
[ERR.] PLE 0022 plexec has timed out.
[INFO] PJM 0083 pjsub Interactive job 497 completed.
```

If a timeout occurs, extend the resource allocation wait time with the --wait-time option of the pjsub command. The wait time depends on the image of the job execution environment. It cannot be decided indiscriminately. Try some values, or use a value 'unlimited' that keeps waiting until the resources are allocated.

The rest of this section describes notes on using each mode.

[Docker mode]

To submit a job to a UDI-specified job execution environment, use the value of the custom resource jobenv for the UDI specification. The administrator will have notified you of this value. Specify the image of the job execution environment in an environment variable.
If "custom-docker" is the jobenv value for UDI specification in docker mode, specify the docker image in an environment variable as follows to submit a job:

- Executing job3.sh on the container instance started from the Docker image file (export format) my-docker.tar

```
$ pjsub -L jobenv=custom-docker -x PJM_JOBENV_DOCKER_IMAGE=/ directory/my-docker.tar job3.sh
```

- Executing job4.sh on the container instance started from the Docker repository my-docker:latest

```
$ pjsub -L jobenv=custom-docker -x PJM_JOBENV_DOCKER_IMAGE=my-docker:latest job4.sh
```

  Specify the repository in the following format for the docker command, etc.

```
[REGISTRY_HOST[:REGISTRY_PORT]/]NAME[:TAG]
```

Whether or not the UDI can be specified depends on the system settings. Therefore, when using UDI specification, check with the administrator about the specification method.

Note

- For UDI specification, the container image must be referenced from compute nodes.
  - For the specifications of an exported tar file, place the file in a user area that can be referenced from compute nodes.
  - For repository specifications, place the repository server of the registered repository at a location that can be accessed from compute nodes via a network.

  Also see "Information" in "E.1.1 In Docker mode" for the container images for the FX server.
- As a security measure, executing a binary with setuid or setgid in a job is prohibited. The su command, etc. cannot be used either.
- The user ID is the same as the user ID on the host OS. However, Technical Computing Suite does not have mapping of user IDs and user names. To use a user name with a program in a job, bind mount the /etc/passwd file of the host OS, or prepare the user name for use within a container.
- Technical Computing Suite uses the following directories as mount points. For this reason, these directories cannot be used as mount points within a container.
  - /etc/opt/FJSVtcs
  - /var/opt/FJSVtcs
  - /usr/libexec/FJSVtcs/krm
  - /var/opt/FJSVtcs/krm/sys/fs/cgroup
  - /run/systemd/journal

- /dev/log

[McKernel mode]

To submit a job to a UDI-specified job execution environment, use the value of the custom resource jobenv for the UDI specification. The administrator will have notified you of this value. Specify the image of the job execution environment in an environment variable.
If "custom-mckernel" is the jobenv value for UDI specification in McKernel mode, specify the McKernel image file in an environment variable as follows to submit a job:

- Executing job.sh on McKernel started from the McKernel image file my-mck.img

```
$ pjsub -L jobenv=custom-mckernel -x PJM_JOBENV_MCKERNEL_IMAGE=/directory/my-mck.img job.sh
```

Whether or not the UDI can be specified depends on the system settings. Therefore, when using UDI specification, check with the administrator about the specification method.

You can specify the following three items when submitting a job in McKernel mode:

- Job script execution environment

  In McKernel mode, you can select host OS or McKernel in the environment variable PJM_JOBENV_MCKERNEL_JOBEXEC as the environment for executing job scripts.

  If "hostlinux" is specified, job scripts are executed on the host OS. If "mckernel" is specified, job scripts are executed on McKernel. If the environment variable is not specified, job scripts are executed on the host OS.

  Settings by the administrator have priority for this setting. Check with the administrator about the validity of the specification in the environment variable.

  To run the job script on host OS, It is necessary to execute the program to be executed via the mcexec command.

  #### See

  For more information about McKernel, see "McKernel User's Guide" available at following URL.

  https://ihkmckernel.readthedocs.io

  - For more information on using the mcexec command and running programs including MPI programs on McKernel, see the chapter "Running Programs".
  - For McKernel configuration, refer to the chapter "Architectural Overview".

- Amount of job memory to allocate to a program running on the host OS

  As described in "1.9 Job Execution Environment," McKernel mode allocates the specified amount of memory when a job is submitted. The allocated memory is divided into an amount for McKernel and an amount for a program running on the host OS (It is referred to here as "for host OS"). The amounts of job memory for McKernel is the one obtained by subtracting the amount of job memory for the host OS from the amount of job memory specified at the job submission. The default job memory amount for the host OS is 128 MiB. You can change the amount with the environment variable PJM_JOBENV_MCKERNEL_JOBMEM. However, it must be at least 128 MiB (See "Note" below). Specify a decimal value in unit of bytes in the environment variable PJM_JOBENV_MCKERNEL_JOBMEM. If any other character string or a value exceeding the total amount of job memory is specified, the job ends with job end code (PJM code) 29.
  The administrator can change the default value for the job memory amount for the host OS. When this is necessary, check with the administrator.

- McKernel boot parameter

  In McKernel mode, you can specify parameters for booting McKernel.
  Boot parameters are items to be set when McKernel OS instance is created (For example, IHK_CPUS or IHK_MEM). For more information, see the following URL.

  https://ihkmckernel.readthedocs.io/en/latest/spec/ihk.html

  Boot parameters are specified at job submission as environment variables with the same name of the setting item.
  To enable the specified boot parameters, the environment variable PJM_JOBENV_MCKERNEL_CUSTOM_ENV must also be

specified at job submission. This value can be any string except an empty string. If you enable the boot parameter specification, memory and CPU are allocated using the values of the boot parameters. Therefore, the boot parameters IHK_CPUS (CPU cores to be allocated) and IHK_MEM (amount of memory to be allocated) must be specified. The -L node-mem option of the pjsub command should not be used to specify the amount of memory.

The following example allocates CPU cores with CPU ID 12 to 23 and 6GB of memory from NUMA node 4 to McKernel.

```
$ pjsub -L jobenv=custom-mckernel \
  -x PJM_JOBENV_MCKERNEL_CUSTOM_ENV=1 -x IHK_CPUS="12-23" -x IHK_MEM="6G@4" job.sh
```

The boot parameters are not checked by the Job Operation Software, but are directly passed to McKernel.

Note

- For UDI specification, the McKernel image file must be referenced from compute nodes.
- Only node-exclusive jobs can run in McKernel mode.

```
$ pjsub -L node=1,jobenv=<Execution Environment Name> ...
```

  In the case of the FX server, the job to which the virtual node is allocated (-L vnode=*N*) cannot be executed because it is a node sharing job.

- Specify the environment variables PJM_JOBENV_MCKERNEL_JOBEXEC and PJM_JOBENV_MCKERNEL_JOBMEM with the -x option of the pjsub command. These environment variables are not valid when specified in a job script.
- cpu affinity of Technical Computing Suite is not reflected by the start of a job script on the host OS or by the start of a program from the mcexec command on McKernel. (cpu affinity is a function to bind a job process to a specific CPU core.) Appropriately specify an option of the mcexec command.
- The pjsig command cannot send the SIGSTOP/SIGCONT signal to processes on McKernel.
- The job memory amount for the host OS (the environment variable PJM_JOBENV_MCKERNEL_JOBMEM) must be 128 MiB or more. Otherwise, the job may not start correctly.
  The following is an example of specifying 1073741824 Byte (1GiB):

```
$ pjsub -x PJM_JOBENV_MCKERNEL_JOBMEM=1073741824 ...
```

  In particular, the default 128MiB of job memory amount for the host OS may not be enough to run an MPI job.
  In McKernel mode, every time one job process is started on McKernel, one delegation process (mcexec) is started on the host OS. mcexec consumes about 10 to 15 MiB of memory per process. Therefore, if many processes start on McKernel, such as running parallel processes in MPI, you need to allocate memory for mcexec on the host OS.
  Specifically, specify (At least 128 MiB) "30MiB(*)+(Number of Processes)x15MiB" or more memory to allocate to the host OS.
  For example, to run an MPI program that is 8 parallel in a node, specify a value of 150MiB(=30MiB+8x15MiB) or greater.
  (*) 30MiB: The amount of basic memory required to start job scripts and MPI processes.

  If the host OS has insufficient job memory, the job has one of the following states. In this case, delete the job, increase the job memory for the host OS, and run the job again.

  a. The item REASON in the job statistical information is set to "LIMIT OVER MEMORY" and the item ASSISTANT CORE MAX MEMORY SIZE (USE) is set to the value equal to the setting of the environment variable PJM_JOBENV_MCKERNEL_JOBMEM.
  b. The item PJM CODE in the job statistical information is set to 21 and transitions to the ERROR state.
  c. The job does not end, and the item RETRY NUM in the job statistical information is set to a value of 1 or greater.

- When specifying McKernel boot parameters, note the following.
  - If the mandatory boot parameters IHK_CPUS and IHK_MEM are not specified, the job ends with PJM code 29.
  - If the memory usage exceeds the amount specified by the boot parameter IHK_MEM, the job ends with PJM code 12.
  - If the boot parameter IHK_KARGS which indicates the kernel arguments is not specified, the McKernel boot options set by the administrator are applied as kernel argument.

- The following functions of the Development Studio are not guaranteed to work.
  - Following MCA parameters of the MPI library

    plm_ple_memory_allocation_policy (specifies the NUMA memory policy)
    plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)

  - High-speed facility on Job Operation Software

  For details of these features, see "MPI User's Guide", "C User's Guide", "C++ User's Guide", "Fortran User's Guide" which are Development Studio manuals.

- Some items in the job statistical information are not output correctly. See "[About job statistical information in McKernel mode]" in "A.2 Output of job statistical information" for details.

[KVM mode]

To submit a job to a UDI-specified job execution environment, use the value of the custom resource jobenv for the UDI specification. The administrator will have notified you of this value. Specify the image of the job execution environment in an environment variable.
If "custom-kvm" is the jobenv value for UDI specification in KVM mode, specify the virtual machine image file in an environment variable as follows to submit a job under the home directory.

- Executing job5.sh on KVM started from the QEMU image file my-kvm.img

```
$ cd ~
$ pjsub -L jobenv=custom-kvm -x PJM_JOBENV_KVM_IMAGE=/directory/my-kvm.img job5.sh
```

Whether or not the UDI can be specified depends on the system settings. Therefore, when using UDI specification, check with the administrator about the specification method.

Note

- For UDI specification, the virtual machine image file must be referenced from compute nodes.
  For additional requirements for the virtual machine image file, see "E.1.3 In KVM mode."
- In the KVM mode, the excess of the memory usage limit configured by job operation software cannot be detected. The behavior when the memory usage exceeds the upper limit on the virtual machine depends on settings of the virtual machine image file.
- In the KVM mode, the location of the files that comprise the job (job script, application, and input/output data files) and the current directory when submitting the job should be under the home directory. If applications need to reference specific modules such as libraries that are not in the guest environment (VM), they should also be in the home directory and available for reference.
- Virtual machine images cannot communicate with each other.
- Only node-exclusive job allocated to one node can run in KVM mode.
  - For FX server

    ```
    $ pjsub -L node=1,jobenv=<Execution Environment Name> ...
    ```

    In the case of the FX server, the job to which the virtual node is assigned (-L vnode=*N*) cannot be executed because it is a node sharing job.

  - For PRIMERGY server

    ```
    $ pjsub -L node=1,jobenv=<Execution Environment Name> ...
    $ pjsub -L vnode=1,jobenv=<Execution Environment Name> -P exec-policy=simplex ...
    ```

  And in the KVM mode, HPC tag address override control function (fhetbo), Power control, Inter-core hardware barrier, and Sector cache are not available.
  Refer to "Job Operation Software End-user's Guide for HPC extensions" for HPC tag address override control function and "Job Operation Software API user's Guide for Power API" for Power control. For Inter-core hardware barrier and Sector cache, refer to the manual for Development Studio.
- The pjshowip, pjrsh, pjpbind, and pjexe commands cannot be used in the KVM mode.

- On a virtual machine, as many CPUs as allocated for a job are numbered in ascending order starting with 0, making a single NUMA node configuration.
- When the pjsig command is used, a signal is sent to the connected ssh process to submit the job to a virtual machine.
- In KVM mode, a job starts after ssh connection to virtual machine. The elapsed time of the job includes the time of ssh connection.
- In the case of the job execution environment where the virtual machine image file can be written to, do not submit multiple KVM mode jobs with the same virtual machine image file. Unexpected events may occur, such as corruption of the virtual machine image file.

See

For the procedure to create an image file of the job execution environment, see "E.1 Creating an Image File for a Job Execution Environment."
To build a job execution environment by yourself, you need to have knowledge about virtualization technology such as the Linux container. If the job execution does not end normally, refer "E.2 Troubleshooting" for the action.

2.4 Checking the Job Status

This section describes how to check the execution status of a job after submitting the job.

2.4.1 Displaying a job list

Users can use the pjstat command to check the states of their own submitted jobs and other related information.

```
$ pjstat

JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE   VNODE  CORE V_MEM
238       job.sh     NM RUN user1    11/17 09:01:41  0001:00:00      12:2x3x2       -      -    -
239       bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00      12:2x3x2       -      -    -
240       step.sh    ST RUN user1    11/17 09:01:42  -               -              -      -    -
241       job2.sh    NM RUN user1    11/17 09:01:42  0001:00:00      2              -      -    -
```

Information

For the meanings of display items, see "A.1 Output of the pjstat and pjstat -v."

Note

- By default, the pjstat command displays only the jobs accessible to the user who executed the command. Some information is shown like "********" for jobs not accessible to the user, even if the jobs should be displayed by the pjstat command with the -A or another option. The administrator may have set jobs to be hidden from users without reference privileges.
- The states of step and bulk jobs are the integrated states of multiple sub jobs. As a result, their job states are not necessarily going to match the state of each of their sub jobs.
  In particular, bulk jobs generate a variety of states. For example, even a bulk job in the RUNOUT state may have some sub jobs still in the QUEUED and RUNNING states.
  To find out the true states of step and bulk jobs, use the -E option of the pjstat command to display sub job states as well. For details, see "a. Displaying sub jobs too."
- The display on your system may differ from the example shown above since the administrator can change the displayed items.
- The -X option of the pjstat command does not support master-worker jobs. (This option displays the node ID and rank number of the compute node where a job is running, that is, in the RUNNING state.) If executed for a master-worker job, the results for the job will be incorrect.

The following examples show the display of job information with the pjstat command.

a. Displaying sub jobs too

By default, the command does not display the sub jobs of a bulk job or step job. To display sub jobs, specify the -E or --expand option.

```
$ pjstat -E

JOB_ID                  JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM  ... (*)
238                     job.sh     NM RUN user1    11/17 09:01:41  0001:00:00  ...
239                     bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00  ...
239[1]                  bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00  ...
239[2]                  bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00  ...
239[3]                  bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00  ...
239[4]                  bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00  ...
239[5]                  bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00  ...
240                     step.sh    ST RUN user1    11/17 09:01:42  -           ...
240_0                   step.sh    ST RUN user1    11/17 09:01:42  0001:00:00  ...
240_1                   step.sh    ST QUE user1    -               0001:00:00  ...
240_2                   step.sh    ST QUE user1    -               0001:00:00  ...
240_3                   step.sh    ST QUE user1    -               0001:00:00  ...
240_4                   step.sh    ST QUE user1    -               0001:00:00  ...
241                     job2.sh    NM RUN user1    11/17 09:01:42  0001:00:00  ...
(*)  The display on the right side is omitted due to space limitations.
```

In the above example, the bulk job (job ID 239) has five sub jobs (sub job IDs 239[1] to 239[5]). All the sub jobs are shown as being executed. The step job (job ID 240) also has five sub jobs (sub job IDs 240_0 to 240_4). The sub job with sub job ID 240_0 is shown as being executed.

Information

Bulk jobs and step jobs have both information corresponding to a job ID and information corresponding to a sub job ID. In this manual, the former is called bulk job summary information or step job summary information to distinguish it.

In the above example, the line with job ID 239 has bulk job summary information, and the line with job ID 240 has step job summary information.

b. Displaying a summary of the number of jobs

Specify the --with-summary option to display the number of jobs submitted by the user executing the pjstat command, as summary information distinguished by state. (If the -A option is also specified, the target is the jobs of all users.)

```
$ pjstat --with-summary

  ACCEPT QUEUED RUNING RUNOUT   HOLD  ERROR SUSPND REJECT   EXIT CANCEL   TOTAL    (*1)
       0      0      4      0      0      0      0      0      0      0       4
s      0      4      8      0      0      0      0      0      0      0      12

JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*2)
238       job.sh     NM RUN user1    11/17 09:01:41  0001:00:00      12:2x3x2      ...
239       bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00      12:2x3x2      ...
240       step.sh    ST RUN user1    11/17 09:01:42  -               -             ...
241       job2.sh    NM RUN user1    11/17 09:01:42  0001:00:00      2             ...

(*1) The summary of the number of jobs
(*2) The display on the right side is omitted due to space limitations.
```

The summary of the number of jobs appears on two lines. The top line adds up, in units of jobs, the number of bulk and step jobs. The lower line (with "s" at beginning of the line) adds up the number of sub jobs in the bulk and step jobs.

Information

The above example shows that two normal jobs, one bulk job (with five sub jobs), and one step job (with five sub jobs), as shown by the itemization in a. above, have been submitted.

- Top line in the summary of the number of jobs
  The total is four because there are two normal jobs, one bulk job, and one step job.
- Bottom line in the summary of the number of jobs
  The total is 12 because there are 2 normal jobs, 5 sub jobs of the bulk job, and 5 sub jobs of the step job.

c. Displaying information on only a specific job

With a job ID or sub job ID specified, the pjstat command displays information on only the specified job. Multiple job IDs can be specified.

```
$ pjstat 238

JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
238        job.sh     NM RUN user1    11/17 09:01:41  0001:00:00      12:2x3x2      ...

$ pjstat '239[1]'

JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
239[1]     bulk.sh    BU RUN user1    11/17 09:01:42  0001:00:00      12:2x3x2      ...
(*)The display on the right side is omitted due to space limitations.
```

Note

If you specify a sub job ID of a bulk job in an argument of the pjstat command, be sure to escape the brackets ("[ ]") with single quotation marks, etc. so that the shell does not process the brackets.

You can specify multiple job IDs and sub job IDs by listing them or specifying a range.

- Listing

```
$ pjstat 238 239                                        (Multiple job IDs)
$ pjstat '239[1]' '239[2]' '239[3]' '239[4]' '239[5]'  (Multiple sub job IDs (bulk job))
$ pjstat 240_0 240_1 240_2 240_3 240_4                  (Multiple sub job IDs (step job))
```

- Specifying a range

  The job IDs, bulk numbers, or step numbers in a range specification have a hyphen between them.

```
$ pjstat 238-240         (Job ID 238, 239, 2401)
$ pjstat '239[1-5]'      (Sub job ID 239[1], 239[2], 239[3], 239[4], 239[5]
$ pjstat 240_0-4         (Sub job ID 240_0, 240_1, 240_2, 240_3, 240_4
```

d. Displaying more detailed job information

You can display more detailed information on each job in the detailed view area by specifying the -v option of the pjstat command.

```
$ pjstat -v

JOB_ID     JOB_NAME   MD ST  USER     GROUP    START_DATE      ELAPSE_TIM ELAPSE_LIM
(*)NODE_REQUIRE    VNODE  CORE V_MEM        V_POL E_POL RANK      LST EC  PC  SN PRI
(*)ACCEPT         RSC_GRP REASON
238        job.sh     NM RUN user1    testgrp  11/17 09:01:41  0000:09:11 0001:00:00
(*)12:2x3x2        -      -    -            -     -     -         RNA 0   0   0  127
(*)11/17 09:01:41 rg001   -
239        bulk.sh    BU RUN user1    testgrp  11/17 09:01:42  -          0001:00:00
(*)12:2x3x2        -      -    -            -     -     -         QUE -   -   -  127
(*)11/17 09:01:41 rg001   -
```

```
240        step.sh    ST RUN user1    testgrp  11/17 09:01:42  -          -
(*)-               -       -    -            -     -     -         -   -   -   -  -
(*)-              -        -
241        job2.sh    NM RUN user1    testgrp  11/17 09:01:42  0000:09:10 0001:00:00
(*)2               -       -    -            -     -     -          RNA 0   0   0  127
(*)11/17 09:01:41 rg001    -

(*) This line continues from the previous line and is actually displayed on the previous line.
```

See

For details on the items displayed when the -v option is specified, see "A.1 Output of the pjstat and pjstat -v."

e. Displaying specific items only

Using the --choose option of the pjstat command, you can display only specific items. In arguments of the --choose option, list the names of items you want displayed. Then, the command displays information in the order listed. For details on the item names that can be specified, see the pjstat(1) man page.

The following example shows the display of the job ID, job name, job model, and job execution start time.

```
$ pjstat --choose jid,jnam,jmdl,sdt

JOB_ID     JOB_NAME   MD START_DATE
238        job.sh     NM 11/17 09:01:41
239        bulk.sh    BU 11/17 09:01:42
240        step.sh    ST 11/17 09:01:42
241        job2.sh    NM 11/17 09:01:42
```

Note

- The administrator may have set that some items not be displayed. Even with these items specified in the --choose option, the command does not display their information.
- You can specify the following item names in the --choose option only: vpol, epol, rankm, and vnid.

f. Displaying only jobs in a specific status.

Using the --filter option, you can display only jobs that satisfy the specified conditions. You can specify the following conditions in the --filter option. For details on the item names that can be specified, see the man page of the pjstat command.

Condition	Format	Description
Value specification	--filter "*item=value*"	Any item that matches the specified value becomes a display target. Example: --filter "jmdl=NM" Display the items where the job model (jmdl) is the normal job (NM). Example: --filter "st=RNA+RUN" Display the jobs in the RUNNING-A or RUNNING state. Note: The "+" symbol indicates the OR condition. To use the "+" symbol as an ordinary character, add a backslash ("\+").
Range specification	--filter "*item=range*"	Any item included in the specified range is a display target. Example: --filter "jid=1-10" Display the jobs whose job IDs are each in a range of 1 to 10. Example: --filter "jid=10-" Display the job whose job IDs are each 10 or higher. Example: --filter "jid=-20" Display the jobs whose job IDs are each 20 or lower. Example: --filter "jid=1-10+21-30"

Condition	Format	Description
		Display the jobs whose job IDs are each in a range of 1 to 10 or 21 to 30.
Character string pattern specification	--filter "*item=character_string_pattern*"	Any item consisting of a character string that matches the specified character string pattern is a display target. You can use the following two types of character string pattern. . (period): Any single character except a line break * (asterisk): Any character string with 0 or more characters except a line break Example: --filter "jnam=jobA*" Display the jobs whose names begin with "jobA". Example: --filter "jnam=job." Display the jobs whose names each consist of "job" followed by one character. Example: --filter "jnam=jobA*+jobB*" Display the jobs whose names begin with "jobA" or "jobB".

Note

If the -A option or --all option is specified, other users' jobs becomes display targets, but hidden information such as user names is ignored even if specified in the --filter option.

g. Displaying information by resource unit or resource group

You can display information by resource unit or resource group by using the --rscunit (--ru) or --rscgrp (--rg) option.

The display examples below show the information for the submission of the following jobs.

```
Job ID 2927: Executed on resource unit: unit1, resource group: group1
Job ID 2928: Executed on resource unit: unit1, resource group: group2
Job ID 2929: Executed on resource unit: unit2, resource group: group1
```

- Displaying information by resource unit
  Specify the --rscunit (--ru) option without arguments. Do not specify the --rscgrp (--rg) option.

```
$ pjstat --rscunit

[ RSCUNIT: unit1 ]
JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
2927       jobA       NM RUN user1    08/06 10:41:45  0001:00:00      2             ...
2928       jobB       NM RUN user1    08/06 10:42:03  0001:00:00      2             ...

[ RSCUNIT: unit2 ]
JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
2929       jobC       NM RUN user1    08/06 10:43:21  0001:00:00      2             ...

(*) The display on the right side is omitted due to space limitations.
```

- Displaying information by resource unit and resource group
  In addition to the --rscgrp (--rg) option, specify the --rscunit (--ru) option without arguments.

```
$ pjstat --rscunit --rscgrp

[ RSCUNIT: unit1 ]
[ RSCGRP: group1 ]
JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
2927      jobA       NM RUN user1    08/06 10:41:45  0001:00:00      2             ...

[ RSCUNIT: unit1 ]
```

```
[ RSCGRP: group2 ]
JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
2928      jobB       NM RUN user1    08/06 10:42:03  0001:00:00      2             ...

[ RSCUNIT: unit2 ]
[ RSCGRP: group1 ]
JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM      NODE_REQUIRE  ... (*)
2929      jobC       NM RUN user1    08/06 10:43:21  0001:00:00      2             ...

(*) The display on the right side is omitted due to space limitations.
```

Information

You can output the displayed data in the form of a list by specifying the --data option. Specify the --rscunit (--ru) and --rscgrp (--rg) options together with this option to add the RSCUNIT (resource unit) and RSCGRP (resource group) fields, respectively, as shown underlined below. This helps determine which resource unit or resource group is pertinent to the contents of the output record.

```
$ pjstat --rscunit unit1 --rscgrp group1 --data
H,RSCUNIT,RSCGRP,,JOB_ID,JOB_NAME,MD,ST,USER,START_DATE,ELAPSE_LIM,NODE_REQUIRE,VNODE,CORE,V_ME
M
,unit1,group1,2927,jobA,NM,RUN,user1, 08/06 10:41:45,0001:00:00,2,-,-,-
```

The pjstat command has more options than described above. For details, see the man page of the pjstat command.

2.4.2 Displaying job statistical information

You can use the -s or -S option of the pjstat command to display job statistical information on a running job. The -S option displays more detailed job statistical information than the -s option.

The following example shows the display of job statistical information.

```
$ pjstat -s

JOB ID                  : 9417
JOB NAME                : script.sh
JOB TYPE                : BATCH
JOB MODEL               : NM
RETRY NUM               : 0
SUB JOB NUM             : -
USER                    : user
GROUP                   : group
RESOURCE UNIT           : rsc_unit
RESOURCE GROUP          : rsc_grp
APRIORITY               : 127
PRIORITY                : 127
SHELL                   : /bin/sh
COMMENT                 :
...
```

Note

- If no job ID is specified, the pjstat command displays job statistical information on all the jobs of the user who executed the command. Job statistical information is displayed with many details, so be aware of this fact when you have numerous jobs.
- Information on the prologue process executed when a job starts is added to the job statistical information for the running job. However, whether the elapsed time of job includes the prologue process depends on the system settings. For details, contact the administrator.
- The update of resource usage information (memory usage and CPU time usage, etc.) among the job statistical information of the running job is delayed in the maximum about ten minutes.

- Since job statistical information includes a lot of information, the administrator may partially limit the job statistical information to be saved in the system. The -s and -S options of the pjstat command display only the information that the administrator has set to be saved.
- The FX server has, besides the CPU core allocated to jobs, a CPU core called the assistant core. The assistant core handles OS interrupt processing and daemon processing that deteriorate job execution performance.
  In jobs, MPI asynchronous communication processing is executed on this assistant core. Therefore, the job statistical information of jobs being executed with the FX server includes, besides the resources allocated to jobs, the assistant core use information related to MPI asynchronous communication processing.

See

For details on the displayed job statistical information, see the man page of the pjstatsinfo(7).

2.4.3 Displaying how a node is used by jobs

Use the pjshowrsc command with the -v 1 option to check the jobs running on a node.

```
$ pjshowrsc -c cluster -v 1
[ CLST: cluster ]
[ NODE: 0x01010010 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32       0      32
      mem       57Gi      0     57Gi

 RUNNING_JOBS: 1022                          <- Job ID of job running on node 0x01010010

[ NODE: 0x0101011 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32      16      16
      mem       57Gi    30Gi    27Gi

 RUNNING_JOBS: 2551                          <- Job ID of job running on node 0x01010011

[ NODE: 0x0101012 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32      16      16
      mem       57Gi    30Gi    27Gi

 RUNNING_JOBS: 2551                          <- Job ID of job running on node 0x01010012
```

Note

The administrator may have restricted access so that other users' jobs cannot be indicated by the "RUNNING_JOBS" line.

Using the pjshowrsc command with the -v 2 option, you can check not only the jobs running on a node but also the usage of custom resources in the node.

```
$ pjshowrsc -c clst1 -v 2
[ CLST: clst1 ]
CUSTOM_RESOURCE                              <- Usage of all custom resources
RSCNAME                       TOTAL       FREE      ALLOC
customrscname1              1000000     300000     700000
customrscname2                    5          5          0
customrscname3/type1      unlimited  unlimited          2
customrscname3/type2      unlimited  unlimited          1

[ NODE: 0x01010010 ]
      RSC     TOTAL    FREE   ALLOC
```

```
      cpu        32       0      32
      mem      57Gi       0    57Gi

 RUNNING_JOBS: 1022                          <- Job ID of job running on node 0x01010010

CUSTOM RESOURCE(PER NODE)                    <- Usage of custom resource in node 0x01010010
RSCNAME                     TOTAL         FREE     ALLOC
customrscname4                  1            0         1

 [ NODE: 0x0101011 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32      16      16
      mem      57Gi    30Gi    27Gi

 RUNNING_JOBS: 2551                          <- Job ID of job running on node 0x01010011

CUSTOM RESOURCE(PER NODE)                    <- Usage of custom resource in node 0x0101011
RSCNAME                     TOTAL         FREE     ALLOC
customrscname4                  1            0         1

 [ NODE: 0x0101012 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32      16      16
      mem      57Gi    30Gi    27Gi

 RUNNING_JOBS: 2552                          <- Job ID of job running on node 0x01010012

CUSTOM RESOURCE(PER NODE)                    <- Usage of custom resource in node 0x0101012
RSCNAME                     TOTAL         FREE     ALLOC
customrscname4                  1            0         1
```

(*1) The names of defined custom resources are displayed at *customrscname1* to *customrscname4*.
(*2) For a custom resource defined with a type, the custom resource name is displayed in the *customrscname/type* format. Also, unlimited is displayed under TOTAL (showing the total quantity of custom resources) and FREE (showing the total quantity of custom resources not allocated to jobs) for the usage.

Using the pjshowrsc command with the -v 3 option, you can check the jobs running on a node, the usage of custom resources in the node, and the jobs using the node as a communication path.

```
$ pjshowrsc -c cluster -v 3
[ CLST: cluster ]
[ NODE: 0x01010010 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32       0      32
      mem      57Gi       0    57Gi

 RUNNING_JOBS: 1022                            <- Job ID of job running on node 0x01010010

CUSTOM RESOURCE(PER NODE)                      <- Usage of custom resource in node 0x01010010
RSCNAME                     TOTAL         FREE     ALLOC
customrscname4                  1            0         1

 JOBS_USING_ROUTE: 1030,1031,1032,1033,1034   <- Job IDs of jobs that use node as communication path

[ NODE: 0x0101011 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32      16      16
      mem      57Gi    30Gi    27Gi

 RUNNING_JOBS: 2551                            <- Job ID of job running on node 0x01010011

CUSTOM RESOURCE(PER NODE)                      <- Usage of custom resource in node 0x0101011
RSCNAME                     TOTAL         FREE     ALLOC
```

```
customrscname4                  1           0         1

 JOBS_USING_ROUTE: 1030,1031,1032,1033,1034   <- Job IDs of jobs that use node as communication path
[ NODE: 0x0101012 ]
      RSC     TOTAL    FREE   ALLOC
      cpu        32      16      16
      mem      57Gi    30Gi    27Gi

 RUNNING_JOBS: 2551                           <- Job ID of job running on node 0x01010012

CUSTOM RESOURCE(PER NODE)                     <- Usage of custom resource in node 0x0101012
RSCNAME                     TOTAL         FREE     ALLOC
customrscname4                  1            0         1

 JOBS_USING_ROUTE: 1030,1031,1032,1033,1034   <- Job IDs of jobs that use node as communication path
```

Information

A job that runs on an FX server may include a job process that performs inter-node communication that goes through a node that is not allocated to the job.
Even if a node in the communication path fails due to a software error, etc., job execution is not affected as long as the interconnect used for communication and the ICC (Interconnect Controller) that controls it are normal. However, if a node becomes incapable of communication due to an ICC failure, etc., all jobs that use the node as a communication path are affected and aborted.

2.4.4 Confirming job end

Use the following methods to find out whether a job has ended.

- Checking the job state by using the pjstat command

  Once a job ends (EXIT, REJECT, or CANCEL state), the pjstat command normally does not display it any longer. Specify the -H or --history option to display only the jobs that have ended.

```
$ pjstat -H

JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM            NODE_REQUIRE ... (*)
1234567    jobname1   NM EXT usrname  01/01 00:00:00  0100:00:00            12:2x3x2     ... (*1)
1234570    jobname1   NM RJT usrname  01/01 00:00:00  0100:00:00            12:2x3x2     ... (*2)
1234590    jobname1   NM CCL usrname  01/01 00:00:00  0100:00:00            12:2x3x2     ... (*3)

(*) The display on the right side is omitted due to space limitations.
(*1) Job in the EXIT state  (*2) Job in the REJECT state  (*3) Job in the CANCEL state
```

Note

- The states of ended jobs are stored only for the length of time set by the administrator. Therefore, even the above operation cannot display the states of the jobs whose storage period has expired.
- Sub jobs that have already ended in a bulk or step job are included in the display by the command with the -H option specified at the end of all of the sub jobs in the same job. Until then, they are included in the display by the command without the -H option specified.
  In the example below, sub job 4_0 of the step job has already ended. The -H option is not specified, so the sub job is displayed because other sub jobs are being executed (job 4 is in the RUNNING state).

```
$ pjstat -E

JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM            NODE_REQUIRE ... (*)
4         t1.sh      ST RUN user1    07/12 13:09:43  -                     -            ...
4_0       t1.sh      ST EXT user1    07/12 13:09:43  0001:00:00            32           ...
4_1       t2.sh      ST RUN user1    07/12 13:50:32  0001:00:00            32           ...
```

```
...
$ pjstat -E -H
JOB_ID    JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM            NODE_REQUIRE ... (*)
1         t1.sh       NM EXT user1   07/05 08:05:11  0001:00:00             32           ...
2         t2.sh       NM EXT user1   07/06 12:30:50  0001:00:00             32           ...

(*) The display on the right side is omitted due to space limitations.
```

- E-mail notification

  If e-mail notification is specified in the -m option of the pjsub command at the job submission time, an e-mail is sent to the user when the job ends.

  The user will always be notified by e-mail, even if the job ends abnormally. For details, see "2.6.1 Referencing job execution results."

- Waiting for job end with the pjwait command

  Users can use the pjwait command to wait for the end of a specific job.

  The pjwait command does not return until the specified job ends.

  The following example shows the submission of three jobs, with the pjwait command waiting for these jobs to end.

```
$ pjsub job1.sh
[INFO] PJM 0000 pjsub Job 5300 submitted.
$ pjsub job2.sh
[INFO] PJM 0000 pjsub Job 5301 submitted.
$ pjsub job3.sh
[INFO] PJM 0000 pjsub Job 5302 submitted.
$ pjwait 5300 5301 5302                          (1)
5300 0 0 -                                       (2)
5301 0 1 -
5302 0 0 -
```

(1) Waiting until all the specified jobs end
(2) Job ID, the job end code, the end status of the job script and the signal number in each ended job.

2.5 Job Operations

This section describes the operations for jobs.

See

The operations described below may not be possible depending on the job state. For details, see "Appendix D Operations on Jobs."

2.5.1 Deleting a job

The operation to cancel a submitted job is called "job deletion." To delete a job, specify the job ID in the pjdel command. Deleting a running job stops the job.

```
$ pjdel jobid [ jobid...]
```

The following message appears when the job is deleted normally.

```
[INFO] PJM 0100 pjdel Accepted job jobid
```

If the specified job does not exist, the following message appears.

```
[ERR.] PJM 0112 pjdel Job non-existing job ID does not exist.
```

In a system using the layered storage system, if the --llio-flush option is specified, the system deletes a job after waiting for the completion of writing from first-layer storage to second-layer storage. However, this wait continues only until the elapsed time limit value for the job

is reached. The elapsed time limit value may be specified with a range (pjsub -L elapse=*min*-*max*). If so, the system waits for the completion of file writing, until the maximum value (*max*) for the elapsed time limit is reached or until the job is terminated to execute the next job.

```
$ pjdel --llio-flush jobid [ jobid...]
```

Note

- If you specify a job although you have no privileges for deleting it, the command operation will be the same as when a non-existing job is specified.
- When a job in the ACCEPT state is deleted, it switches to the REJECT state. When a job in a state other than the ACCEPT state is deleted, it switches to the CANCEL state.
- To perform job deletion that involves the aborting of a running prologue script (RUNNING-P state) or a running epilogue script (RUNNING-E state), specify the --enforce option. Note also that if you delete a job with the --enforce option before epilogue script execution, the epilogue script will be left unexecuted.
  In some cases, the administrator configures the settings so that users cannot specify the --enforce option. Check the pjacl command output item, execute pjdel(--enforce).
- The pjdel command returns after requesting the job operation management function to delete a job. Job deletion processing including the wait for the completion of file writing is processed asynchronously with the return of the pjdel command.
- Deleting a large number of queued jobs (QUEUED state) that were submitted incorrectly can result in a large number of job statistical information files and a large amount of job history information displayed by the -H option of the pjstat command. To avoid this, specify the following options:

  --no-stats

  If the job being deleted is in the QUEUED state, suppress the output of the job statistical information file (.stats file) for that job. When this option is specified, suppress the output of the job statistical information file, even for jobs that is submitted with the --stats or --STATS options of the pjsub command.

  --no-history

  If the job being deleted is in the QUEUED state, suppress the output of that job to the job history information that is output by the -H option of the pjstat command. Note that the administrator may have disabled the --no-history option. Check the output item of the pjacl command, execute pjdel (--no-history).

  Regardless of the above options, the administrator may have configured to suppress the output of the job statistical files or job history information. Contact administrator for configuration information.

2.5.2 Sending a signal to a job

Users can send signals to running jobs (RUNNING state) by using the pjsig command.

```
$ pjsig -s signal job ID
```

Users can specify a signal with a signal name (e.g., SIGHUP) or a signal number (1 to 64).

The following example sends the signal SIGKILL (signal number 9) to a job.

```
$ pjsig -s 9 1                              <- Specifies a signal by a signal number.
[INFO] PJM 0700 pjsig Accepted job 1.

$ pjsig -s SIGKILL 2                        <- Specifies a signal by a signal name.
[INFO] PJM 0700 pjsig Accepted job 2.
```

2.5.3 Holding a job and canceling the hold

When the pjdel command deletes a running job, the job ends. To execute the job again, the user has to submit the job again.

Using the pjhold command will abort the job and discard the execution results, but the job remains submitted. This is called "holding the job," and this state is called the HOLD state.

To cancel the HOLD state, use the pjrls command. This command schedules the job again to execute it from the beginning.

If the system stops during job execution, hold the job once, and then cancel the hold after operation resumes. This saves the effort of submitting the job again.

Note

To perform job holding that involves the aborting of a running prologue script (RUNNING-P state) or a running epilogue script (RUNNING-E state), specify the --enforce option. Note also that if you hold a job with the --enforce option before epilogue script execution, the epilogue script will be left unexecuted.
In some cases, the administrator configures the settings so that users cannot specify the --enforce option. Check the pjacl command output item, execute pjhold(--enforce).

The following example holds a job and cancels the hold.

```
$ pjhold 1 2
[INFO] PJM 0300 pjhold Accepted job 1.
[INFO] PJM 0300 pjhold Accepted job 2.

$ pjstat -v 1-2
JOB_ID     JOB_NAME   MD ST  USER   ...     REASON
1          jobname1   NM HLD user1  ...     user1
2          jobname2   NM HLD user1  ...     user1

$ pjrls 1 2
[INFO] PJM 0400 pjrls Job 1 released.
[INFO] PJM 0400 pjrls Job 2 released.

$ pjstat -v 1-2
JOB_ID     JOB_NAME   MD ST  USER   ...  LST ...  REASON
1          jobname1   NM QUE user1  ...  HLD ...  -
2          jobname2   NM QUE user1  ...  HLD ...  -
```

In a system using the layered storage system, if the --llio-flush option is specified, the system holds a job after waiting for the completion of writing from first-layer storage to second-layer storage. However, this wait continues only until the elapsed time limit value for the job is reached. The elapsed time limit value may be specified with a range (pjsub -L elapse=*min*-*max*). If so, the system waits for the completion of file writing, until the maximum value (*max*) for the elapsed time limit is reached or until the job is terminated to execute the next job.

```
$ pjhold --llio-flush jobid [ jobid...]
```

Note

The pjhold command returns after requesting the job operation management function to hold a job. Job hold processing including the wait for the completion of file writing is processed asynchronously with the return of the pjhold command.

2.5.4 Changing job parameters

After a user submits a job, the user can use the pjalter command to change the following job parameters.

- -L option
  - Elapsed time limit value (elapse)
  - Resource unit for executing a job (rscunit)
  - Resource group for executing a job (rscgrp)

```
$ pjalter [-L | --rsc-list] rscname=value jobid
```

- -p option
  - Priorities among jobs of the same user (priority)

```
$ pjalter -p priority jobid
```

Note

- Job parameters can be changed when the job status is QUEUED, HOLD, or ERROR.
  When permitted by the administrator in the job operation management function settings for jobs executed on an FX server, you can even change the elapsed time limit value for a job in the RUNNING state. However, a change made with end-user privileges can only shorten the elapsed time limit value. The changed value cannot be smaller than the time that has already elapsed.
- The user might not been permitted to execute as for the pjalter command.
- If the sub job ID of the bulk job is specified, it is possible only to change the elapsed time limit value.
- If you are changing the resource unit or resource group on which a job runs, the job ACL function must be set to the following for the user who submitted the job (the parentheses denote the output of the pjacl command) :
  - Job submission is allowed for the destination resource unit or resource group ('execute pjsub' is set to enable).
  - The options specified at the time of job submission are also allowed in the resource unit or resource group to which it is being changed ('execute pjsub(*opt*)' is set to 'enable').
  - The resource amount specified at the time of job submission is within the upper and lower limits ('upper' and 'lower' in 'pjsub option parameters') in the resource unit or resource group to which it is being changed.
  - If the job uses custom resources, the resource unit or resource group to which it is being changed must also have the custom resources defined (The custom resources to be use are displayed in 'pjsub option parameters').

  To check the above settings, specify the resource unit or resource group you want to change.in the pjacl command.

```
$ pjacl --rscunit resourceunit
$ pjacl --rscgrp resourcegroup
```

- None of the parameters can change jobs whose elapsed time limit value is specified with a range. (See "2.3.2.3 Specifying the elapsed time limit value for a job.")
- The parameter is changed, the job is rescheduled at the times set by the administrator, and the changed parameters are applied. The information displayed by the pjstat command is updated as parameters are applied to the job.

2.6 Checking Job Results

This section describes how to check job execution results.

2.6.1 Referencing job execution results

The standard output and standard error output of a job are created as separate files in the current directory at the job submission time. However, if the job is a bulk job or step job, the standard output and standard error output are output as separate files for each sub job.

The following table lists the default file names for the output destination. You can change the file name when submitting a job. (See "2.3.2.10 Specifying the standard output and standard error output files of a batch job.")

Job output	Output destination file name
Standard output	*<job name>.<jobID>*.out However, for a bulk job or step job, the job ID part is replaced by a sub job ID. Example 1: The job name is job.sh, and the job ID is 123456. job.sh.123456.out Example 2: The job name is bulkjob.sh, and the sub job IDs of the bulk job are 123456[0], 123456[1], and so on.

Job output	Output destination file name
	bulkjob.sh.123456[0].out, bulkjob.sh.123456[1].out, ... Example 3: The job name is stepjob.sh, and the sub job IDs of the step job are 123456_0, 123456_1, and so on. stepjob.sh.123456_0.out, stepjob.sh.123456_1.out,...
Standard error output	*<job name>.<job ID>*.err However, for a bulk job or step job, the job ID part is replaced by a sub job ID. Example 1: The job name is job.sh, and the job ID is 123456. job.sh.123456.err Example 2: The job name is bulkjob.sh, and the sub job IDs of the bulk job are 123456[0], 123456[1], and so on. bulkjob.sh.123456[0].err, bulkjob.sh.123456[1].err, ... Example 3: The job name is stepjob.sh, and the sub job IDs of the step job are 123456_0, 123456_1, and so on. stepjob.sh.123456_0.err, stepjob.sh.123456_1.err, ...

Note

- If the job name begins with a single-byte numeric character, "J" is added to the beginning of the output file name.
- The job name part of an output file name (including the letter "J" added to the beginning) consists of up to 63 characters.
- For a job submitted from the standard input and not by a job script, the job name part is "STDIN".
- If the standard output and standard error output for a bulk job or step job have the same file name specified, the file will contain a mixture of output from each sub job.
- When you execute the mpiexec command in a job that runs on the FX server, its standard output and standard error output are written to the file defined by the item 'mpiexec-*' of the job ACL function (see "2.2.2 Checking restriction information") unless you specify otherwise with the mpiexec command option.
  For more information about the standard output and standard error output of the mpiexec command, see "2.3.6.9 Standard output/standard error output of the mpiexec command [FX]" and the Development Studio manual"MPI User's Guide."

The Job Operation Software may output the following messages to the standard output or standard error output of the job.

```
[Priority] PJM nnnn message text              (Standard error output)
[Priority] PLE nnnn message text              (Standard output or standard error output)
[Priority] PLE nnnn plexec message text       (Standard output or standard error output)
```

Format	Meaning
[*Priority*]	Indicates the significance of the message by showing [ERR.], [WARN], or [INFO].
PJM	Indicates a message of the job manager function. Since the administrator can set whether to output this message, it may not always be output for the occurrence of the event.
PLE	Indicates a message of a function of the parallel execution environment.
plexec	Indicates a message of the plexec command that is an internal function of the parallel execution environment.
nnnn	Message ID that is a 4-digit integer

See

For details on these messages, see "Messages in job outputs" in the "Chapter 3 Command Reference for End-user" of the "Job Operation Software Command Reference" manual.

If an error occurs in a job, the user who executed the pjsub command for the job is notified of the error by e-mail. If job acceptance is rejected, such notification is not sent.

The following example shows e-mail notification.

```
Subject: PJM job: 6814. error.                    (1)

Job name:       pjmtest.sh                        (2)
Job owner:      user1                             (3)
Mail sent at:   Fri Jun 11 21:33:55 JST 2010      (4)


Reason: Node down.                                (5)
```

(1) Indication of abnormal end of the job of job ID 6814
(2) Job name
(3) Name of the user who submitted the job
(4) E-mail send time
(5) Cause of the abnormal end

See

- For details on the messages in the e-mail notification, see "Appendix B Notification Message Related to Job Execution."
- If the job execution in the job exection environment does not end normally, refer "E.2 Troubleshooting" for the action.

Note

The mail is send to the account on the compute cluster management node, if --mail-list option of the pjsub command is not specified. The mail delivery to the user depends on the system design. For details, ask the administrator.

Information

If an error occurs in a job, the reason is output to job statistical information.

2.6.2 Outputting job statistical information

If you specify the -s or -S option of the pjsub command when submitting a job, job statistical information will be output to either of the following files. One is a file (file name: *<job name>.<job ID>*.stats) in the current directory at the job submission time, and the other is the file specified in the --spath option. For details, see "A.2 Output of job statistical information" or man page pjstatsinfo(7).

Otherwise, you can reference the job statistical information on ended jobs by specifying the -H option and the -s or -S option of the pjstat command.

```
$ pjstat -H                                                                  (1)
...
JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM         NODE_REQUIRE  ... (*)
1234567    jobname1   NM EXT usrname  01/01 00:00:00  0100:00:00         12:2x3x2      ...
...

$ pjstat -H  -s 1234567                                                      (2)
...

(*) The display on the right side is omitted due to space limitations.
```

(1) Checks for the job IDs of ended jobs.
(2) Specifies a job ID to display job statistical information. (The -S option displays details.)

For details on the displayed items and their meanings, see the man page of the pjstat command.

Note

- The pjstat command does not display the states of any jobs whose storage period has expired, so users cannot reference these jobs. The administrator sets the storage period of a job state.
- The statistical information output in job execution on FX servers differs slightly from that of PRIMERGY servers.
- Information the prologue and epilogue processes set by the administrator is added to statistical information of the job.
  However, whether the elapsed time of job includes the prologue and epilogue processes depends on the system settings. For details, contact the administrator.
- The job statistical information of jobs executed with the FX server includes, besides the resources allocated to jobs, the assistant core use information related to the MPI asynchronous communication processing.

Information

A job that is aborted because of a system problem, such as a node failure, may be automatically re-executed. (See "2.3.2.7 Specifying automatically re-execution for a job.") In such cases, the re-execution results are the job completion results.

2.7 Effects of Node Failures on Jobs

This section describes the effect of a node failure during job execution on the job.

Table 2.37 Effects of compute node failures on jobs

Target	Re-execution allowed (*)	Re-execution not allowed (*) or interactive job
Job	The job is canceled and transitions to the QUEUED state. After computer resources are assigned, the job is re-executed.	The job is canceled and terminated.
Job statistical information file (*.stats file)	Job statistical information from the job cancellation time is not output. The job re-execution results are output.	Job statistical information from the job cancellation time is output.
Standard output (*.out file) and standard error output (*.err file) of the job	The results until the job was canceled and the job re-execution results are output.	The results until the job was canceled are output.
File output by the job	The results until the job was canceled are output. If an existing file at the re-execution time has the same name, the handling depends on the job file operation specifications.	The results until the job was canceled are output.

(*) The --restart or --norestart option specified at the job submission time or the job ACL function settings specify whether job re-execution is permitted.

Appendix A Job Information

This appendix describes the information of jobs output with the pjstat command and the pjsub command.

A.1 Output of the pjstat and pjstat -v

The pjstat command outputs job information.

```
$ pjstat
JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM            NODE_REQUIRE     (*)
VNODE  CORE V_MEM
XXXXXXXXXX XXXXXXXXXX XX XXX XXXXXXXX MM/DD hh:mm:ss  hhhh:mm:ss-hhhh:mm:ss nnnnnn:XXxYYxZZ  (*)
nnnnnn nnn  nnnnnnnnnnMiB
...
(*) This line continues to the next line.
```

Specify the -v option to display additional information.

```
$ pjstat -v
JOB_ID     JOB_NAME   MD ST  USER     GROUP   START_DATE      ELAPSE_TIM ELAPSE_LIM       (*)
NODE_REQUIRE     VNODE  CORE V_MEM        V_POL E_POL RANK       LST EC  PC  SN PRI       (*)
ACCEPT         RSC_GRP  REASON
XXXXXXXXXX XXXXXXXXXX XX XXX XXXXXXXX XXXXXXX MM/DD hh:mm:ss< hhhh:mm:ss hhhh:mm:ss-hhhh:mm:ss (*)
nnnnnn:XXxYYxZZ  nnnnnn nnn nnnnnnnnnnMiB XXXXX XXXXX XXXXXXX    XXX XXX XXX XX XXX       (*)
MM/DD hh:mm:ss XXXXXXXX XXXXXXXXXXXXXXXX

(*) This line continues to the next line.
```

The following table lists the displayed information and its meanings.

Information

- Items with [-v] appended in the Name column are displayed when the -v option is specified.
- Bulk jobs and step jobs have information corresponding to the job ID and information corresponding to the subjob ID. The former in particular is called summary information of the bulk or step job.

Table A.1 Output items of the pjstat command

Name	Description
JOB_ID	Job ID (10-digit decimal number) For the sub job, the sub job ID is output. (*jobID*[*BulkNumber*] *or jobID_StepNumber*)
JOB_NAME	Job name (Only first 10 characters)
MD	Job model NM: Normal job ST: Step job BU: Bulk job MW: Master-worker job
ST	Current processing state of the job ACC: Accepted job submission CCL: Exit by job cancelation ERR: In fixed state due to an error EXT: Completed of the job termination processing HLD: Holding status by user operation QUE: Waiting for job execution RJT: Rejected job submission RNA: Acquiring resources required for job execution

Name	Description
	RNE: In the processing of the epilogue RNO: Waiting for completion of job termination processing RNP: In the processing of the prologue RSM: In the processing of the resuming RUN: Executing job SPD: Suspended SPP: In the processing of the suspending
USER	Name of user name who executes job (Only first 8 characters) If the user is not registered in the OS, the user ID is output.
GROUP [-v]	Group name of user executing job (Only first 8 characters) For the summary information of the step job, the group name that was specified when the first subjob for the step job was submitted is output. If the group is not registered in the OS, the group ID is output. If there is no running sub job, the information of the sub job to be executed next is output.
START_DATE	Planned execution start time or execution start time - (*MM*/*DD hh*:*mm*) Planned execution start time. The planned execution start time is enclosed in parentheses. - (*YYYY*/*MM*/*DD*) Planned execution start time (after one year or later) - (*MM*/*DD hh*:*mm*)# If the planned execution start time is after the scheduling period, "#" is added after the time. In this case, the planned execution start time is after displayed time. - *MM*/*DD hh*:*mm*:*ss* Actual start time of the job - (*MM*/*DD hh*:*mm*)< or *MM*/*DD hh*:*mm*:*ss*< As for jobs to which backfill is applied, "<" is added after the time. - (*MM*/*DD hh*:*mm*)@ or *MM*/*DD hh*:*mm*:*ss*@ As for jobs to which the execution start time is specified, "@" is added after the time. For the summary information of the step job, the information of the running sub job is output. If there is no running sub job, the information of the sub job to be executed next is output. The administrator may have set the display of the planned execution start time as follows: - Instead of the time, output the character string decided by the administrator. - Do not output some or all of the following additional characters after the planned execution start time: "#", "<", and "@".
ELAPSE_TIM [-v]	Elapsed time limit "*hhhh*:*mm*:*ss*" When column overflow occurs, output is executed with *ss* omitted. The job elapsed time does not include the suspend period. For the summary information of the bulk job or step job, "-" is output.
ELAPSE_LIM	Elapsed time limit There are two display formats. - *time* (*hhhh*:*mm*:*ss*) hhhh:mm:ss is the longest possible time for the execution of a job. - *time1*-*time2* (*hhhh*:*mm*:*ss*-*hhhh*:*mm*:*ss*) This range indicates how long a job can be executed, provided that there are free resources, when the elapsed time of job execution has passed "*time1*" but not yet reached "*time2*."

Name	Description
	When column overflow occurs, output is executed with *ss* omitted. For the summary information of the step job, "-" is output. For the summary information of the bulk job, "-" is output when each sub job has a different elapsed time limit value.
NODE_ REQUIRE	- FX server Shape and number of nodes when job is submitted: "*nnnnnn*:*XX*x*YY*x*ZZ*" When format above is too short to hold information, only node shape is output. - PRIMERGY server Number of nodes when the job is submitted: "*nnnnnn*" If nothing is specified, "-" is output. For the summary information of the step job, "-" is output.
VNODE	Number of virtual nodes "*nnnnnn*" For the following jobs, "-" is output: - Job on an FX server - Node allocated job on a PRIMERGY server This item may have been set by the administrator to not be displayed.
CORE	Number of CPU cores per virtual node "*nnn*" For the following jobs, "-" is output: - Job on an FX server - Node allocated job on a PRIMERGY server This item may have been set by the administrator to not be displayed.
V_MEM	Amount of memory per virtual node "*nnnnnnnnnn*MiB" This is the value of the parameter vnode-mem or mem that is specified when the job is submitted. For a job with core-mem specified, the value is multiplied by the number of CPU cores per virtual node. core-mem is the memory amount per CPU core. For the following jobs, "-" is output: - Job on an FX server - Node allocated job on a PRIMERGY server This item may have been set by the administrator to not be displayed.
V_POL [PG] [-v]	Virtual node placement policy A_PCK : Absolutely PACK PACK : PACK A_UPK : Absolutely UNPACK UPCK : UNPACK For the following jobs, "-" is output: - Job on an FX server - Node allocated job on a PRIMERGY server - The summary information of the step job This item may have been set by the administrator to not be displayed.
E_POL [PG] [-v]	Execution mode policy SHARE : SHARE SMPLX : SIMPLEX For the following jobs, "-" is output: - Job on an FX server - Node allocated job on a PRIMERGY server - The summary information of the step job This item may have been set by the administrator to not be displayed.
RANK [PG] [-v]	Allocation methods with a rank map bynode : rank-map-bynode bychip : rank-map-bychip

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>For the following jobs, "-" is output:
- Job on an FX server
- Node allocated job on a PRIMERGY server
- The summary information of the step job

This item may have been set by the administrator to not be displayed.</td></tr>
<tr><td>LST
[-v]</td><td>Last processing state of the job (prior state from which the "current processing state of the job" is shifted)
For the meaning of the value, see the ST item.</td></tr>
<tr><td>EC
[-v]</td><td>Exit code of a job script
For the summary information of the bulk job or step job, "-" is output.</td></tr>
<tr><td>PC
[-v]</td><td>Job end code (PJM code)
Code that indicates the job manager processing result of the job execution
For the summary information of the bulk job or step job, "-" is output.
The meanings of the codes are as follows.
For some job end codes, a message corresponding to the code is output to the standard error output of the job. For details on the meaning and action for the message, see "Messages in job outputs" in "Chapter 3 Command Reference for End-user" of the "Job Operation Software Command Reference" manual. For details on the actions for the job end codes marked with (*), contact the administrator.
<table>
<tr><th>Job end code (PJM code)</th><th>Meaning</th></tr>
<tr><td>0</td><td>Normal end of job</td></tr>
<tr><td>1</td><td>CANCEL by the pjdel command operated by a user</td></tr>
<tr><td>2</td><td>REJECT by the judgment of the job acceptation.
A pjsub command error occurs.</td></tr>
<tr><td>3 (*)</td><td>Execution rejection by the job manager exit function
The job is not executed.</td></tr>
<tr><td>4</td><td>HOLD by the pjhold command operated by a user</td></tr>
<tr><td>6</td><td>Cancellation due to evaluation of step job dependency expression.
The job is not executed.</td></tr>
<tr><td>7</td><td>Cancellation due to deadline forcibly</td></tr>
<tr><td>8 (*)</td><td>CANCEL by the job manager exit function
The job is not executed.</td></tr>
<tr><td>9</td><td>Specification of norestart job, so job is EXIT when rebuild Job information.</td></tr>
<tr><td>11</td><td>Job execution timeout due to elapsed time limit violation</td></tr>
<tr><td>12</td><td>Forced termination due to use of excessive amount of memory</td></tr>
<tr><td>16</td><td>Job termination due to an current directory or standard input/standard output/ standard error output file inaccessible.</td></tr>
<tr><td>18</td><td>Job end due to the execution of a subsequent job or the start of the deadline schedule, when the job has been running for longer than the minimum executable time.

If the cause is the former, the REASON item becomes "ANOTHER JOB STARTED." If it is the latter, the item becomes "DEADLINE SCHEDULE STARTED."</td></tr>
<tr><td>20</td><td>Node failure</td></tr>
<tr><td>21</td><td>Failure in job script execution</td></tr>
<tr><td>22</td><td>ICC error</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Name</th><th colspan="2">Description</th></tr>
<tr><td></td><td></td><td>Note: This also applies to cases where it is judged that even changing the communication path when a Tofu interconnect link goes down cannot make job execution continue (see "2.3.2.12 Specifying the operation when a Tofu interconnect link is down [FX]").</td></tr>
<tr><td></td><td>23</td><td>Termination due to OOM Killer</td></tr>
<tr><td></td><td>25</td><td>Failure in HA</td></tr>
<tr><td></td><td>26 (*)</td><td>Error in the prologue or epilogue process</td></tr>
<tr><td></td><td>27 (*)</td><td>Error in the resource management exit process</td></tr>
<tr><td></td><td>28</td><td>Job execution environment error</td></tr>
<tr><td></td><td>29</td><td>Invalid job execution environment information specified</td></tr>
<tr><td></td><td>30</td><td>Abort due to a suspend or resume process failure</td></tr>
<tr><td></td><td>100 (*)</td><td>Internal error in the job manager</td></tr>
<tr><td></td><td>120 (*)</td><td>Internal error in the job scheduler</td></tr>
<tr><td></td><td>140 (*)</td><td>Internal error in job resource management</td></tr>
<tr><td></td><td>160 (*)</td><td>Internal error in the Tofu library [FX]
The job is not executed.</td></tr>
<tr><td></td><td>180</td><td>Internal error in the layered storage system</td></tr>
<tr><td>SN
[-v]</td><td colspan="2">Signal number
For the summary information of the bulk job or step job, "-" is output.</td></tr>
<tr><td>PRI
[-v]</td><td colspan="2">Job priority (0 to 255)
A higher number indicates higher priority.
For the summary information of the step job, "-" is output.</td></tr>
<tr><td>ACCEPT
[-v]</td><td colspan="2">Job submission date and time "MM/DD hh:mm:ss"</td></tr>
<tr><td>RSC_GRP
[-v]</td><td colspan="2">Resource group when job is submitted</td></tr>
<tr><td>REASON
[-v]</td><td colspan="2">Error message
Message corresponding to result code for some processing of a job regardless of whether job is executed
For the summary information of the bulk job or step job, "-" is output.
The meanings of output messages are as follows.
<table>
<tr><th>Error message</th><th>Meaning</th></tr>
<tr><td>-</td><td>No error</td></tr>
<tr><td>MM/DD hh:mm DELAY</td><td>Job execution is scheduled after the specified start time.</td></tr>
<tr><td>ANOTHER JOB STARTED</td><td>Job end due to the execution of a subsequent job, when the job has been running for longer than the minimum executable time</td></tr>
<tr><td>DEADLINE SCHEDULE STARTED</td><td>Job end due to the start of the deadline schedule, when the job has been running for longer than the minimum executable time</td></tr>
<tr><td>ELAPSE LIMIT EXCEEDED</td><td>Exceeded limit of the elapse time</td></tr>
<tr><td>FILE IO ERROR</td><td>Current directory at which the job is submitted is not accessible</td></tr>
<tr><td>GATE CHECK</td><td>Cancellation by the job manager exit function</td></tr>
<tr><td>IMPOSSIBLE SCHED</td><td>Impossible scheduling</td></tr>
<tr><td>NSUFF CPU</td><td>Insufficient number of physical CPUs</td></tr>
<tr><td>INSUFF MEMORY</td><td>Insufficient physical memory</td></tr>
</table></td></tr>
</table>

Name	Description	
	INSUFF NODE	Insufficient number of physical nodes
	INSUFF *CustomResourceName*	Insufficient custom resource defined with the resource name *CustomResourceName*
	INTERNAL ERROR	Internal error
	INVALID HOSTFILE	Invalid host file specified in the rank-map-hostfile parameter of the pjsub command [FX]
	LIMIT OVER MEMORY	Exceeded limit of memory in job execution
	LOST COMM	All-to-all communication among parallel processes not guaranteed
	NO CURRENT DIR	Unable to access current directory or standard input/standard output/standard error output file at user job submission
	NOT EXIST *CustomResourceName*	No custom resource is defined with the resource name *CustomResourceName*.
	RESUME FAIL	Resume failed
	RSCGRP NOT EXIST	Resource group does not exist
	RSCGRP STOP	Resource group stopped
	RSCUNIT NOT EXIST	Resource unit does not exist
	RSCUNIT STOP	Resource unit stopped
	RUNLIMIT EXCEED	Exceeded limit of the number of simultaneous job executions
	SUSPEND FAIL	Suspend failed
	USELIMIT EXCEED	Waiting for execution due to limitation of simultaneous node or CPU core usage
	USER NOT EXIST	Job execution user does not exist in the system
	WAIT SCHED	Exempt from scheduling because the limit on the number of jobs subject to scheduling has been reached
	Other character strings	- Message specified in the --reason option of the pjdel, pjhold, or pmsuspend command - Message that is set by the administrator using the job manager exit function, job scheduler exit function, or job resource manager exit function On the job held by the pjhold command and the job suspended by the pmsuspend command, the string is output as "*username*: *reasonmessage*". If the --reason option is omitted, the string is output as "*username*:" (*) "*username*" is the name of user who executed the pjhold command.

Specify the --with-summary or --summary option of the pjstat command to display the number of jobs by state. The following table lists the displayed information and its meanings.

```
$ pjstat --with-summary

  ACCEPT QUEUED RUNING RUNOUT   HOLD  ERROR SUSPND REJECT   EXIT CANCEL   TOTAL
  nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnnn
s nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnn nnnnnnn

JOB_ID     JOB_NAME   MD ST  USER     START_DATE      ELAPSE_LIM           NODE_REQUIRE     (*)
```

```
VNODE  CORE V_MEM
XXXXXXXXXX XXXXXXXXXX XX XXX XXXXXXXX MM/DD hh:mm:ss  hhhh:mm:ss-hhhh:mm:ss nnnnnn:XXxYYxZZ  (*)
nnnnnn nnn  nnnnnnnnnnMiB

(*) This line continues to the next line.
```

Table A.2 Output items of the pjstat command (number of jobs by state)

Name	Description
ACCEPT	Number of jobs waiting for job acceptance
QUEUED	Number of jobs waiting for job execution
RUNING	Number of jobs being executed
RUNOUT	Number of jobs waiting job end
HOLD	Number of jobs in fixed state due to a user
ERROR	Number of jobs in fixed state due to an error
SUSPND	Number of jobs during suspend period (state is SUSPEND, SUSPENDED, or RESUME) This item may have been set by the administrator to not be displayed.
REJECT	Number of jobs that were refused acceptance
EXIT	Number of jobs that have ended
CANCEL	Number of canceled jobs
TOTAL	Total number of jobs in each displayed state

A.2 Output of job statistical information

Job statistical information is output when the -s or -S option of the pjstat command and pjsub command is specified.

Note

The administrator can change the items and output order of output job statistical information. For this reason, the output job statistical information may differ from the following examples.

- pjstat -s output example

```
$ pjstat -s
[Job Statistical Information]
 JOB ID                      : 100
 JOB NAME                    : job.sh
 JOB TYPE                    : BATCH
 JOB MODEL                   : NM
 ...
 <omitted>
 ...
 START BULKNO                : -
 END BULKNO                  : -
```

- pjstat -S output example

```
$ pjstat -S
[Job Statistical Information]
 JOB ID                      : 100
 JOB NAME                    : job.sh
 JOB TYPE                    : BATCH
 JOB MODEL                   : NM
 ...
 START BULKNO                : -
```

```
 END BULKNO                  : -

[Node Statistical Information]             (*)
 VNODE ID                    : -
 NODE ID                     : 0x00014DBF
 ...
 CPU BITMAP (USE)            : 0xF
 RANK NO                     : 0
```

(*) Information for each node or virtual node

- pjsub -s output example (job statistical information file)

```
Job Statistical Information

 JOB ID                      : 100
 JOB NAME                    : job.sh
 JOB TYPE                    : BATCH
 ...
 <omitted>
 ...
 START BULKNO                : -
 END BULKNO                  : -
```

- pjsub -S output example (job statistical information file)

```
Job Statistical Information

 JOB ID                      : 100
 JOB NAME                    : job.sh
 JOB TYPE                    : BATCH
 ...
 <omitted>
 ...
 START BULKNO                : -
 END BULKNO                  : -

--------------------------------------------------------------------------------
Node Statistical Information                    (*)

 NODE ID                     : 0x00014DBE
 TOFU COORDINATE             : (23,17,16)
 ...
 CPU BITMAP (USE)            : 0xF
 RANK NO                     : 0

--------------------------------------------------------------------------------
Node Statistical Information                    (*)

 NODE ID                     : 0x00014DBF
 ...
 CPU BITMAP (USE)            : 0xF
 RANK NO                     : 1
```

(*) Statistic information for each node or virtual node

See

For details on the items of job statistical information output by the pjstat command or pjsub command, see the pjstatsinfo(7) man page.

Note

- PERF COUNT n item of job statistical information
  The PERF COUNT 1 to PERF COUNT 9 items in job statistical information are related to FX server CPU performance information. These items have a value of 0 in the following cases:
  - The profiler function or runtime information output function of Development Studio is used in a job.
  - The xospastop command is executed in a job (For the xospastop command, see "2.1.1.7 Creating a job that uses PAPI library or strace command [FX].")

  The PERF COUNT 1 to PERF COUNT 9 items in job statistical information are related to FX server CPU performance information. For interactive jobs, these items have a value of "-".

  The PERF COUNT *n* item cannot be obtained on an MPI processing system other than Development Studio. The calculation is not included in job statistical information.

  You can calculate job performance information from the PERF COUNT *n* item. For details, see "A.3 Calculating Job Performance Information [FX]."
  If a job uses the Fujitsu profiler, the PMU counter (Performance Monitoring Unit Counter) stops collection. From the point that it stopped, the latest information periodically collected by the job resource manager is set in this item. You can check the FJ PROFILER item to see whether or not a job used the Fujitsu profiler.
- The job statistical information of jobs executed with the FX server include not only the calculation of the processing with CPU cores allocated to jobs, but also the calculation of the MPI asynchronous communication processing processed by the assistant core.
  Therefore, with FX server jobs, item CPU NUM(ALLOC) and CPU NUM(USE) may increase to a size larger than item CPU NUM(REQUIRE). Also, item USER CPU TIME (USE), SYSTEM CPU TIME (USE), and CPU TIME(TOTAL) include the time of CPUs executed by the assistant core.
- The minimum measurement unit of CPU time spent for job execution on the FX server is 10ms(10000000ns.If the CPU time for job execution is less than the minimum measurement unit, it is rounded to 0 as the job statistical information.
- Average node power consumption of FX server outputs two types of (Estimated) and (actual measurement), respectively.
  The average node power consumption (Estimated) is estimated by hardware from based on the number of CPU instructions issued and other information. This value does not take into account variations in power consumption due to difference of computing and I/O node devices which are assistant core, PCI Express and so on.
  The average node power consumption (actual measurement) is collected by hardware from an electric energy measurement element. This value takes into account variations in power consumption due to difference of computing and I/O node devices which are assistant core, PCI Express and so on. This value also varies due to individual differences and data patterns to be processed.
  When same job is executed on different nodes, the average node power consumption (Estimated) is same, however the average node power consumption (actual measurement) may be different. There may be a difference of +/- 40% between these two types of node average power consumption.
  Maximum node power consumption, minimum node power consumption, and node power consumption also outputs two types of (Estimated) and (actual measurement), but above notes are the same.

[About job statistical information in McKernel mode]

For McKernel mode job statistics, note the following.

- The environment for running McKernel mode jobs is both the host OS and McKernel (Refer to "McKernel mode" in "1.9 Job Execution Environment"). Therefore, McKernel mode job statistical information are basically for both the host OS and McKernel, but some of them are for the environment shown below.

Table A.3 Environments subject to job statistical information in McKernel mode

Job statistical informatino	Environments subject to job statistical information
USER CPU TIME (USE)	McKernel
SYSTEM CPU TIME (USE)	McKernel
CPU TIME (TOTAL)	McKernel
MAX MEMORY SIZE (USE)	McKernel

Job statistical informatino	Environments subject to job statistical information
ASSISTANT CORE (USE)	Host OS
ASSISTANT CORE USER CPU TIME (USE)	Host OS
ASSISTANT CORE SYSTEM CPU TIME (USE)	Host OS
ASSISTANT CORE MAX MEMORY SIZE (USE)	Host OS
FJ PROFILER	McKernel
SECTOR CACHE	McKernel
INTRA NODE BARRIER	McKernel
UTILIZATION INFO OF POWER API	McKernel
CPU BITMAP (USE)	Host OS (*1)
PERF COUNT 1 - PERF COUNT 9	McKernel (*2)

(*1)
Item CPU BITMAP (USE) outputs the CPU ID bitmap information based on the CPU numbering scheme on the host OS.

(*2)
The items PERF COUNT 1 - PERF COUNT 9 show the value of CPU statistical information used on McKernel. These items did not include CPU statistical information used by processes started on the host OS.

- The CPU time in job statistical information of jobs running in McKernel mode is usually greater than the CPU time measured by the user calling the getrusage() system call in the job process. This is because the CPU time of the auxiliary program (mcexec) called from McKernel to invoke the job process is also counted.
- If the McKernel boot parameters are specified (see "2.3.8 Specifying a job execution environment"), the job statistical information described below may not be correct.
  - CPU BITMAP (USE)
  - USER CPU TIME (USE)
  - SYSTEM CPU TIME(USE)
  - CPU TIME (TOTAL)
  - MEMORY SIZE(ALLOC)
  - MAX MEMORY SIZE(USE)
  - CPU NUM (USE)

[About job statistical information in KVM mode]

In KVM mode, the job operation software cannot retrieve some resource information used within the virtual machine. Therefore, the job statistical information for these are output as follows.

Table A.4 Job statistical information of job in the KVM mode

Item	Description
USER CPU TIME (USE)	This item represents information about the ssh process connecting to the virtual machine.
SYSTEM CPU TIME (USE)	This item represents information about the ssh process connecting to the virtual machine.
CPU TIME (TOTAL)	This item represents information about the ssh process connecting to the virtual machine.
MAX MEMORY SIZE (USE)	This item represents the maximum memory usage of the QEMU process and the ssh process connecting to the virtual machine.

Item	Description
CPU NUM (USE)	This item represents the CPU usage of the ssh process connecting to the virtual machine.
ASSISTANT CORE (USE)	For FX server, "0" is always output. For PRIMERGY server, "-" is always output regardless of KVM mode.
ASSISTANT CORE USER CPU TIME (USE)	For FX server, "0" is always output. For PRIMERGY server, "-" is always output regardless of KVM mode.
ASSISTANT CORE SYSTEM CPU TIME (USE)	For FX server, "0" is always output. For PRIMERGY server, "-" is always output regardless of KVM mode.
PERF COUNT 1 - PERF COUNT 9	"-" is always output.

A.3 Calculating Job Performance Information [FX]

You can calculate job-related performance information on the FX server from the job statistical information as follows.

Table A.5 How to calculate job performance information

Performance Information	Value
Number of execution cycles	SUM(PERF COUNT 1)
Number of floating-point instruction operations	SUM(PERF COUNT 2)+SUM(PERF COUNT 3)x4
Number of memory read requests	SUM(PERF COUNT 4)/12
Number of memory write requests	SUM(PERF COUNT 5)/12
Number of sleep cycles	SUM(PERF COUNT 6)

PERF COUNT *N*: The item of the job statistical information

SUM(perf) : Total value of the perf item that is output in the job statistical information for each node

Appendix B Notification Message Related to Job Execution

If job execution does not end normally and if the user gave an instruction to be notified of job start and job end, the user receives a detailed notification by e-mail.

The following table lists the contents of the Reason line in e-mail notification. For details of the notification by e-mail, see "2.6.1 Referencing job execution results."

Table B.1 E-mail notification contents

Message	Meaning	State of job	Action
Reason: Internal error: *code*.	Internal error occurred An internal code is output at *code*.	Transition to ERROR state	Contact the administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message.
Reason: Node down.	A node failure occurred while being executing the job or requesting the job.	Transition to QUEUED or termination	Contact the administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message. Since the aborted job is automatically re-executed, the user does not need to take any action. However, for jobs that automatically re-execution is disabled, the job must be resubmitted.
Reason: Unable to analyze pjm *code*.	An invalid code was received from the job resource management function, job scheduler function, or the layered storage system.	Transition to ERROR state	Contact the administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message.
Reason: Fail to write the jobinfo file. *Details* path:*pathname*	It failed in the output of statistical information. A cause is output at *Details*. The path could not write *pathname* is displayed.	Termination of job	Contact the administrator, if the message "internal error" is output at *Details*. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message. In other than the above, check the write permission of *pathname*. Also, show the job ID to check with the administrator whether statistical information on the job can be obtained from the information recorded in the system. If the pmdumpjobinfo command is able to obtain statistical information on the job, the administrator should consider providing information to the user.
Reason: Fail to write the jobinfo record. *Details* path:*pathname*			

Message	Meaning	State of job	Action
Fail to get user name or group name.	Information on the group or user was not able to be acquired.	Transition to ERROR state	The existence of the group or user is confirmed.
Host file is not correct.	The rank-map-hostfile parameter of the pjsub command does not specify the correct host file.	Transition to ERROR state	Confirm the REASON with execute the pjstat command -v option. If output in the REASON is the INVALID HOSTFILE, review the host file. In other case, contact the administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message.
Fail to access current directory.	When the job submitted, the current directory was inaccessible.	Transition to ERROR state	The existence of the current directory is confirmed.
The current directory does not exist or have permission. path: *pathname*.	At job submission, the current directory or the standard input/ standard output/standard error output file was inaccessible. The *pathname* is a path name of the current directory.	Termination of job	Confirm the existence of the current directory at job submission and, the path name of the standard input/standard output/ standard error output file. Or, confirm whether the user submitting the job has access privileges for them.
Job *jobid* was canceled due to the suspend processing failure of the system.	Suspend processing failed and the job was aborted.	Transition to QUEUED or termination	Since the aborted job is automatically re-executed, the user does not need to take any action. However, for jobs that automatically re-execution is disabled, the job must be resubmitted.
Job *jobid* canceled by pjdel command from *username*.	The pjdel command executed by the user *username* deleted the job.	Delete	No action is necessary.
Job *jobid* canceled, because of norestart job.	The job is not re-executed because automatically re-execution is disabled.	Delete	To execute the job, resubmit it.
The job was canceled because of the deadline period.	The job was interrupted according to the deadline schedule.	Transition to QUEUED state	If there are free resources, the job is re-executed. If not, the job will be re-executed after the deadline schedule ends.
CPU time limit exceeded.	The CPU utilization time exceeded the limit value.	Termination of job	Check the resource limit value for the job.
Elapse time limit exceeded.	The elapsed time exceeded the limit value.	Termination of job	Check the resource limit value for the job.
Memory size limit exceeded.	The memory usage exceeded the limit value.	Termination of job	Check the resource limit value for the job.
Killed by OOM killer.	OOM Killer in the OS forcibly terminated the job.	Termination of job	Because OOM Killer can occur due to the behavior of the job, review the job in two ways. If these are not the case, contact your administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu

Message	Meaning	State of job	Action
			systems engineer (SE) with the collected data together with the output message. 1. Restricting the memory allocation to specific NUMA nodes If you specify MPOL_BIND in the set_mempolicy system call to restrict a process's memory acquisition to a specific NUMA node, OOM Killer kills the process when it runs out of memory on that NUMA node. In this case, stop restricting the memory acquisition target NUMA nodes or reduce the amount of memory acquisition for the job. 2. Using up to near the amount of memory resource(*) for the job of the compute node Depending on the behavior of the job, if the job memory is fragmented at that time, or if the memory resource for the job is less than the memory usage specified when the job was submitted, you may be notified as "Killed by OOM killer."(PJM CODE 23) instead of "Memory size limit exceeded."(PJM CODE 12). In this case, reduce the amount of memory acquisition for the job. (*) The amount of memory resource displayed by the pjshowrsc command
Job *jobid* was deleted by system factor.	The job was deleted due to a system factor.	Delete	Contact the administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message.
Abnormal end.	An abnormality occurred in job execution.	Termination of job	Contact the administrator. The administrator shall collect investigation data according to "Job Operation Software Administrator's Guide for Maintenance," and then contact a Fujitsu systems engineer (SE) with the collected data together with the output message.
Job *jobid* is started.	The job started.	-	No action is necessary. This notification message is output only if the -m b option is specified in the pjsub command.
Job *jobid* is completed.	The job ended normally.	-	No action is necessary. This notification message is output only if the -m e option is specified in the pjsub command.

Message	Meaning	State of job	Action
Job *jobid* restarted.	The job was re-executed.	-	No action is necessary. This notification message is output only if the -m r option is specified in the pjsub command.
Another job started.	After running for longer than the minimum executable time, the job ended due to the execution of a subsequent job.	-	No action is necessary. This notification message is output only if the -m e option is specified in the pjsub command.
Deadline schedule started.	After running for longer than the minimum executable time, the job ended due to the start of the deadline schedule.	-	No action is necessary. This notification message is output only if the -m e option is specified in the pjsub command.
Reason: Gate check. or Reason: *Message set by the administrator.*	The job was aborted by the job manager exit function, job scheduler exit function, or job resource manager exit function, as set by the administrator.	Delete	Check with the administrator about the reason for this abort.
Reason: *rscname*=*value* is less than the lower limit (*limit-value*).	The value of *rscname*=*value* specified at job submission is less than the lower limit value (*limit-value*).	Termination of job	Using the pjacl command, check the lower limit value of the resource *rscname*. Specify a value that is not less than that.
Reason: *rscname*=*value* is greater than the upper limit (*limit-value*).	The value of *rscname*=*value* specified at job submission is more than the upper limit value (*limit-value*).	Termination of job	Using the pjacl command, check the upper limit value of the resource *rscname*. Specify a value that is not more than that.

Appendix C Executing programs of MPI processing system other than Development Studio

C.1 Notes and Execution Examples for Each MPI Processing System

This appendix describes examples of executing MPI programs of the MPI processing system other than Development Studio on the Job Operation Software and the notes on executing them. These MPI programs include Intel MPI, OpenMPI, MPICH, or Platform MPI.

[NOTE]

- To execute an MPI process on a node or virtual node, use the pjrsh command which is supplied with the Job Operation Software instead of rsh and ssh command.
  The pjrsh command gives a standard input to the MPI process when a standard input in the mpiexec process is used by the Intel MPI (Hydra) etc.
  However, there is a possibility to which the pjrsh's transfer performance is deteriorated compared with rsh command and ssh command when the volume of data of a standard input exceeds about 8MB. In such case, please do not use a standard input in the mpiexec process but read the file by each MPI process.

- Specify the node resources to be allocated to a job as shown below.

  - To allocate node resources in units of nodes

```
pjsub -L "node=N" ...
```

  - To allocate node resources in units of virtual nodes
    Specify Absolutely UNPACK (abs-unpack) as the virtual node placement policy.

```
pjsub -L "vnode=N,vnode-core=M" -P "vn-policy=abs-unpack" ...
or
pjsub -L "vnode=N(core=M)" -P "vn-policy=abs-unpack" ...
```

- For an MPI program that uses all the logical CPUs in a CPU core, execute the pjacl command and confirm that "all" is the value of the setting item define assign-logical-cpu of the job ACL function.

- To execute an MPI program by specifying the host name, operate as follows.

  a. Executing on FX servers
     Use the pjshowip command.
     The pjshowip command is executed in a job script and outputs the allocated node list (IP addresses of each rank number) to the standard output.

  b. Executing on PRIMERGY servers
     Use the environment variable PJM_O_NODEINF.
     An environment variable PJM_O_NODEINF is set by the Job Operation Software. It indicates the path of the file storing the node list (a list of IP addresses) which is a list of the nodes allocated to the job.

  For details, refer to the following example.

- To delete an MPI job, send a signal of the MPI processing system that executes the cleanup process (such as resource releases), and then delete the job. If a job is deleted without sending the signal for executing the cleanup process, it may have some impact on operations. For example, the subsequent job may not be executed.

  For signals that execute the cleanup process, check the specifications of MPI processing systems. Generally, the signal SIGINT or SIGTERM executes the cleanup process.

[Execution examples]

The examples of execution on each MPI processing system are shown below. Note that the version listed is an example and is not a limitation.

For details of the way of executing the MPI programs and the version dependency of MPI, confirm the specifications of each MPI processing system.

- Intel MPI 2021 [PG]

  Submitting the job script that executes the following a series of operations enables execution of MPI programs of Intel MPI on the virtual node of the job.

  [Using the Scalable Process Management System (Hydra)]

  1. Setting environment variables

     Set Hydra environment variables as follows.

     a. I_MPI_HYDRA_BOOTSTRAP=rsh

     b. I_MPI_HYDRA_BOOTSTRAP_EXEC=/bin/pjrsh

     c. I_MPI_HYDRA_HOST_FILE="${PJM_O_NODEINF}"

     Henceforth, the pjrsh command is used instead of rsh and ssh commands.

  2. MPI program execution

     Execute the MPI program that was created with Intel MPI. To execute the program, use the mpiexec.hydra command.

  [Example] Job script that executes a process parallel MPI program 'a.out' (when using Hydra)

```
...
export I_MPI_PERHOST=1
export I_MPI_HYDRA_BOOTSTRAP=rsh                        1.a.
export I_MPI_HYDRA_BOOTSTRAP_EXEC=/bin/pjrsh            1.b.
export I_MPI_HYDRA_HOST_FILE="${PJM_O_NODEINF}"         1.c.
mpiexec.hydra -n 4 a.out                                2.
...
```

Note

MPI programs of Intel MPI cannot be executed on the virtual node of the job using Multipurpose Daemon (MPD).

Information

The Job Operation Software has prepared the wrapper command mpiexec.tcs_intel for the mpiexec.hydra command, to provide Intel MPI with an execution view similar to that for Development Studio MPI.
For examples of job scripts that use the wrapper command mpiexec.tcs_intel, see "C.4 MPI Program Execution with the Wrapper Command mpiexec.tcs_intel [PG]."

- MPICH 3.4.2 [PG]

  Submitting the job script that executes the following a series of operations enables execution of MPI programs of MPICH on the virtual node of the job.

  [Using the Scalable Process Management System (Hydra)]

  1. Setting environment variables
     Set Hydra environment variables as follows.

     a. HYDRA_BOOTSTRAP=rsh

     b. HYDRA_BOOTSTRAP_EXEC=/bin/pjrsh

     c. HYDRA_HOST_FILE="${PJM_O_NODEINF}"

     Henceforth, the pjrsh command is used instead of rsh and ssh commands.

2. MPI program execution
   Execute the MPI program that was created with MPICH. To execute the program, use the mpiexec.hydra command.

[Example] Job script that executes a process parallel MPI program (a.out) (when using Hydra)

```
...
export HYDRA_BOOTSTRAP=rsh                                  1.a.
export HYDRA_BOOTSTRAP_EXEC=/bin/pjrsh                      1.b.
export HYDRA_HOST_FILE="${PJM_O_NODEINF}"                   1.c.
mpiexec.hydra -n 4 a.out                                    2.
...
```

- OpenMPI 4.1.5

As shown below, MPI program of OpenMPI can be executed on a node or virtual node of the job.

Note

When you build and install the openmpi-4.0.0 source files from the OpenMPI download site, install them on a shared file system that can be referenced with the same path on the login node and all compute nodes.

1. Add the following to the installation destination of OpenMPI, etc/openmpi-mca-params.conf file.

```
plm_rsh_agent=/bin/pjrsh
```

2. Executing an MPI program

   a. Executing on FX servers
      Specify the node list file which was output by the pjshowip command with the -machinefile option of mpiexec in the job script.
      [Example] Job script that executes a process parallel MPI program (a.out)

```
xospastop
pjshowip > ./iplist_"${PJM_JOBID}"
mpiexec -n 4 -machinefile ./iplist_"${PJM_JOBID}" --map-by node:SPAN a.out
rm -f ./iplist_"${PJM_JOBID}"
```

   b. Executing on PRIMERGY server
      Specify the node list file ${PJM_O_NODEINF} with the -machinefile option of mpiexec in the job script.
      [Example] Job script that executes a process parallel MPI program (a.out)

```
mpiexec -n 4 -machinefile "${PJM_O_NODEINF}" --map-by node:SPAN a.out
```

Information

About warning message

If the following warning message is output, the mpiexec option --mca btl_openib_allow_ib 1 may suppress warning output.

```
You can override this policy by setting the btl_openib_allow_ib MCA parameter to true.
```

For details, see the information published by the OpenMPI project.

- Platform MPI 9.1.4 [PG]

Submitting the job script that executes the following series of operations enables execution of MPI programs of Platform MPI on the virtual node of the job.

1. Set /bin/pjrsh to the environment variable MPI_REMSH of Platform MPI. By setting it, pjrsh command is used instead of rsh and ssh command.

2. Executing an MPI program

   Execute an MPI program created on Platform MPI. Then, specify the node list file ${PJM_O_NODEINF} with the -hostfile option.

[Example] Job script that executes a process parallel MPI program (a.out)

```
...
export MPI_REMSH=/bin/pjrsh                              1.
mpirun -np 4 -hostfile "${PJM_O_NODEINF}" a.out          2.
...
```

C.2 Binding CPU Resources to Processes [PG]

To execute an MPI program of the MPI processing system other than Development Studio using the Job Operation Software on the PRIMERGY servers, use the pjpbind command so that CPU resources (CPU cores or logical CPUs) can be bound in the same way as with Development Studio.

The pjpbind command binds CPU resources to its processes and then executes the program specified in an argument. At this time, from the CPU resources in the nodes allocated to the job, CPU resources that are not bound to any process are preferentially selected and then bound to processes.
For details on the pjpbind command, see the man manual.

[Example] Job script executing a hybrid parallel MPI program (a.out)

```
#PJM -L "vnode=2"
#PJM -L "vnode-core=8"
export OMP_NUM_THREADS=8
mpiexec -n 2 pjpbind a.out
```

Like in the above example, you can bind CPU resources to threads when executing a multithreaded MPI program with the pjpbind command.
To do so, specify the number of threads in the environment variable PARALLEL or OMP_NUM_THREADS. If these environment variables are specified concurrently, the environment variable PARALLEL has priority. For a program using a runtime library that can interpret the environment variable GOMP_CPU_AFFINITY of GNU gcc, the pjpbind command can bind CPU resources to threads without duplication between multiple processes on one node.

With the --disable-thread option specified in the pjpbind command, the command automatically binds CPU resources to processes but not to threads in the program to be executed.

You can change the pattern of CPU resource binding by setting the following environment variables when using the pjpbind command.

Table C.1 Environment variables related to binding CPU cores

Environment variable name	Description
PLE_MPI_PIN_DOMAIN	Specifies "omp" or integer *n*. - "omp" specified The number of CPU resources that are bound to processes is the same as the value specified in the environment variable PARALLEL or OMP_NUM_THREADS. If both of the environment variables are specified concurrently, the environment variable PARALLEL has priority. If both of the environment variables are omitted, the operation is the same as with integer value 1 specified. - Integer *n* specified The number of CPU resources bound to processes is *n*. Either CPU cores or logical CPUs are bound, which is determined by the setting of the environment variable PLE_MPI_PIN_CELL. If the specified value is neither "omp" nor integer *n* or if this environment variable has no setting, the operation is the same as with "omp" specified.
PLE_MPI_PIN_CELL	Specifies "core" or "unit". - "core" specified CPU resources are bound to processes for each CPU core.

Environment variable name	Description

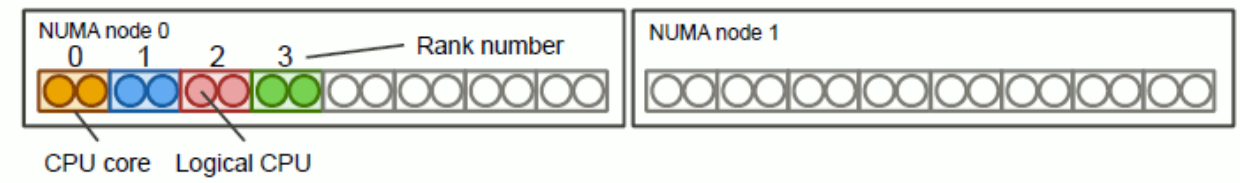
	- "unit" specified
	CPU resources are bound to processes for each logical CPU.
	
	If the specified value is neither "core" nor "unit" or if this environment variable has no setting, the operation is as follows.
	
	- Same operation as with "unit" specified
	  - The environment variable PLE_MPI_PIN_DOMAIN is set.
	  - The environment variable PLE_MPI_PIN_DOMAIN is not set, and multiple job processes are bound to one CPU core on the node where the pjpbind command started.
	- Same operation as with "core" specified
	  - The environment variable PLE_MPI_PIN_DOMAIN is not set, and only one job process is bound to one CPU core on the node where the pjpbind command started.
	  - The environment does not have Intel(R) Hyper-Threading Technology enabled.

PLE_MPI_PIN_ORDER	Specifies "range", "compact", or "scatter". - "range" specified CPU resources are bound to processes in ascending order of CPU ID, so the CPU resource bound to the current process has a higher ID than that for the previous process. - "compact" specified CPU resources are bound such that the CPU resource bound to the current process is close to that of the previous process. - "scatter" specified CPU resources are bound such that the CPU resource bound to the current process is far from that of the previous process. In any case, in a job, the starting point of CPU resource binding in a cycle with the above order is the CPU resource with the smallest CPU ID. Either CPU cores or logical CPUs are bound, which is determined by the setting of the environment variable PLE_MPI_PIN_CELL. If the specified value is none of "range", "compact", and "scatter" or if this environment variable has no setting, the operation is the same as with "scatter" specified.

The following examples show CPU resource binding using these environment variables.

[Example 1]

```
...
export PLE_MPI_PIN_CELL=core
export PLE_MPI_PIN_ORDER=compact
mpiexec -n 4 pjpbind a.out
```

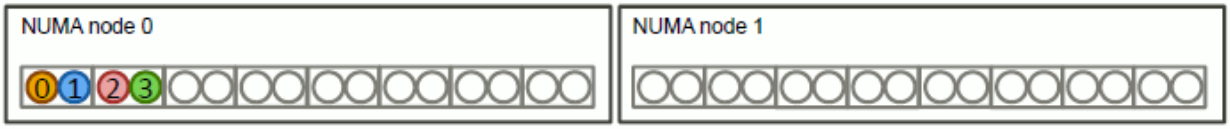


[Example 2]

```
...
export PLE_MPI_PIN_CELL=core
export PLE_MPI_PIN_ORDER=scatter
mpiexec -n 4 pjpbind a.out
```

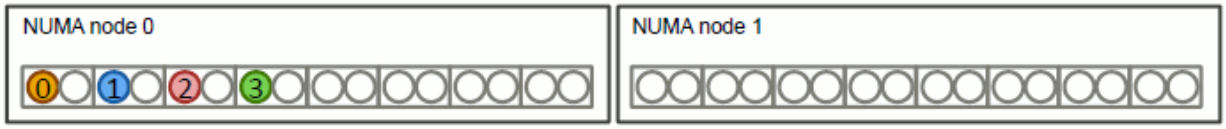
[Example 3]

```
...
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=compact
mpiexec -n 4 pjpbind a.out
```



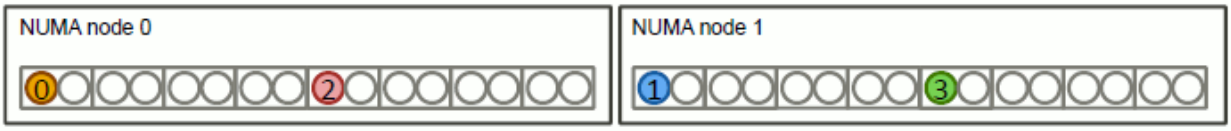
[Example 4]

```
...
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=range
mpiexec -n 4 pjpbind a.out
```



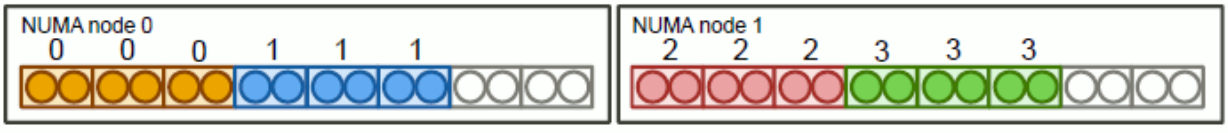
[Example 5]

```
...
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=scatter
mpiexec -n 4 pjpbind a.out
```



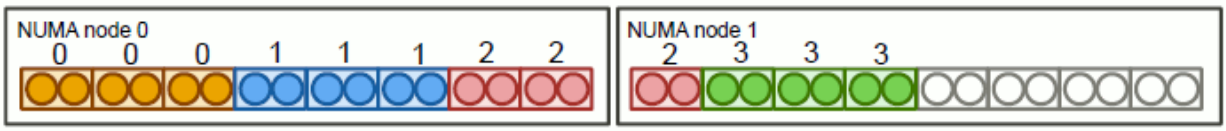
[Example 6]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=omp
export PLE_MPI_PIN_CELL=core
export PLE_MPI_PIN_ORDER=compact
mpiexec -n 4 pjpbind a.out
```



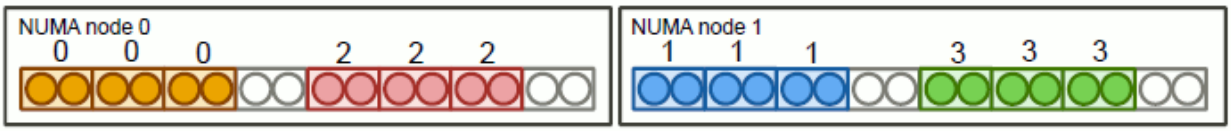
[Example 7]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=omp
export PLE_MPI_PIN_CELL=core
export PLE_MPI_PIN_ORDER=range
mpiexec -n 4 pjpbind a.out
```



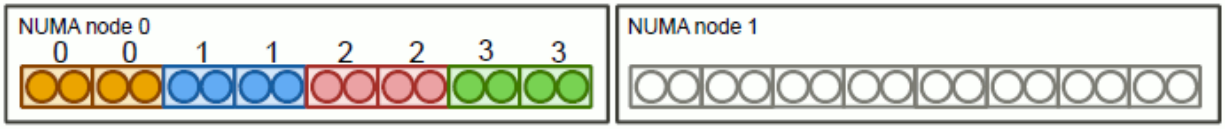
[Example 8]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=omp
export PLE_MPI_PIN_CELL=core
export PLE_MPI_PIN_ORDER=scatter
mpiexec -n 4 pjpbind a.out
```



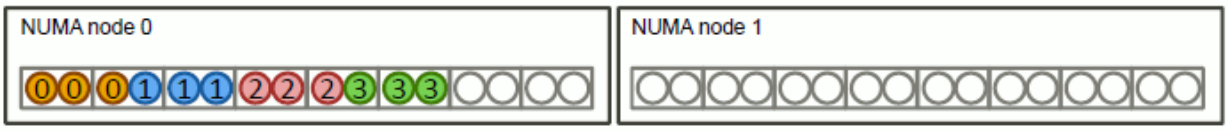
[Example 9]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=2
export PLE_MPI_PIN_CELL=core
export PLE_MPI_PIN_ORDER=compact
mpiexec -n 4 pjpbind a.out
```



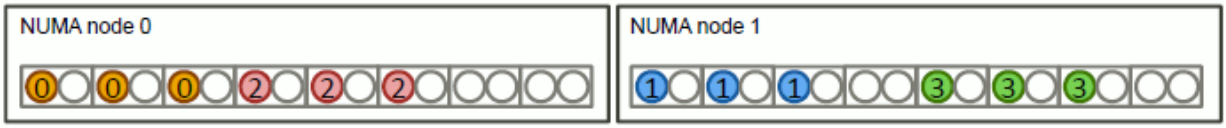
[Example 10]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=omp
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=compact
mpiexec -n 4 pjpbind a.out
```



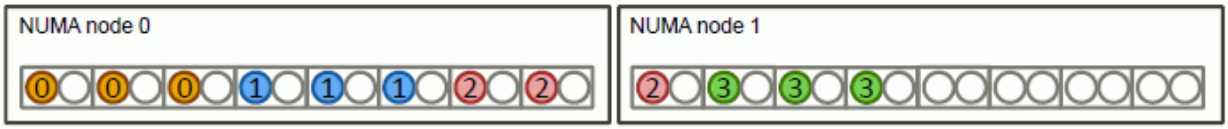
[Example 11]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=omp
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=scatter
mpiexec -n 4 pjpbind a.out
```



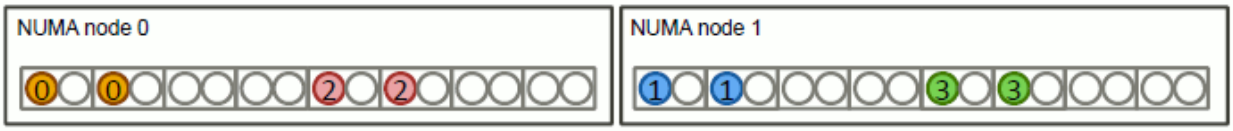
[Example 12]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=omp
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=range
mpiexec -n 4 pjpbind a.out
```



[Example 13]

```
...
export OMP_NUM_THREADS=3
export PLE_MPI_PIN_DOMAIN=2
export PLE_MPI_PIN_CELL=unit
export PLE_MPI_PIN_ORDER=scatter
mpiexec -n 4 pjpbind a.out
```



C.3 Setting the NUMA Memory Allocation Policy [PG]

For the execution of an MPI program of an MPI processing system other than Development Studio, you can set the NUMA memory allocation policy by using the environment variable PLE_MEMORY_ALLOCATION_POLICY.

Note

- The environment variable PLE_MEMORY_ALLOCATION_POLICY is enabled only when the pjpbind command is executed.
- To set the NUMA memory allocation policy, the numactl package must be installed on the compute node. Contact your administrator to install the numactl package.

On compute nodes in the NUMA architecture, job execution performance may be unstable or degraded because of the speed of access to NUMA memory. The NUMA memory allocation policy is a memory allocation method for reducing this effect.

The environment variable PLE_MEMORY_ALLOCATION_POLICY can have the same values as in the MCA parameter plm_ple_memory_allocation_policy of Development Studio MPI. For details on the values that can be set in the MCA parameter plm_ple_memory_allocation_policy and the NUMA memory allocation policy, see "Setting Values for the NUMA Memory Allocation Policy" in "MPI User's Guide," which is a Development Studio.

C.4 MPI Program Execution with the Wrapper Command mpiexec.tcs_intel [PG]

The following examples show the execution of job scripts using the wrapper command mpiexec.tcs_intel for the mpiexec.hydra command of Intel MPI.

Note

To use the wrapper command mpiexec.tcs_intel, the environment variable PATH in the job must include the installation directory of the mpiexec.hydra command of Intel MPI.

[Example 1] Job script executing a process parallel MPI program (a.out)

```
#!/bin/sh
#PJM -L "node=4"
mpiexec.tcs_intel a.out
```

[Example 2] Job script executing a flat parallel MPI program (a.out)

```
#!/bin/sh
#PJM -L "node=4"
#PJM --mpi "proc=64"
mpiexec.tcs_intel a.out
```

[Example 3] Job script executing a hybrid parallel MPI program (a.out)

```
#!/bin/sh
#PJM -L "node=4"
#PJM --mpi "proc=16"
export OMP_NUM_THREADS=4
mpiexec.tcs_intel a.out
```

The pjpbind command, which is described in "C.2 Binding CPU Resources to Processes [PG]," is internally called to bind CPU resources to processes when an MPI program is executed using the wrapper command mpiexec.tcs_intel.

You can specify a pjpbind command option in the environment variable PLE_I_MPI_PJPBIND_OPT.
To prevent CPU resources from being bound automatically by the pjpbind command in the wrapper command mpiexec.tcs_intel, specify "disable" in the environment variable PLE_I_MPI_PJPBIND.

Table C.2 Environment variables of the wrapper command mpiexec.tcs_intel

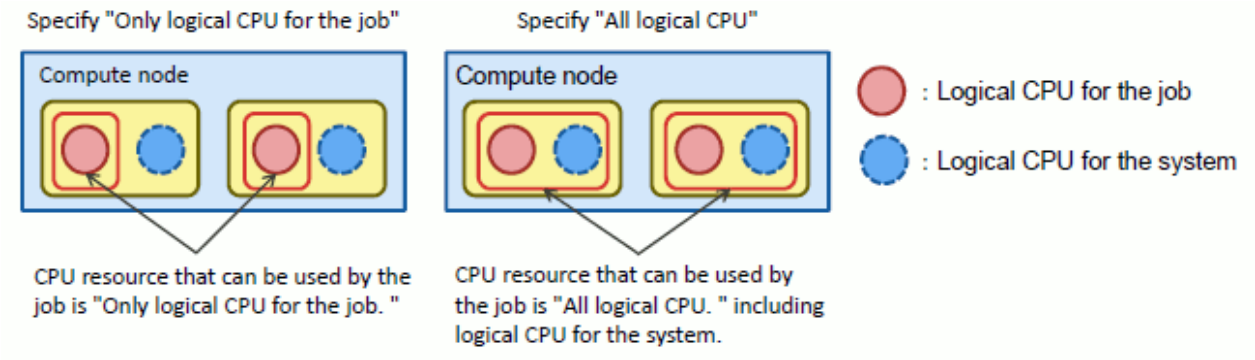
Environment variable name	Description
PLE_I_MPI_PJPBIND_OPT	Specifies a pjpbind command option with a character string. Example: export PLE_I_MPI_PJPBIND_OPT="--disable-thread" This environment variable is enabled only when the environment variable PLE_I_MPI_PJPBIND has no setting or has "enable" specified for it.
PLE_I_MPI_PJPBIND	Specifies "enable" or "disable". - enable The pjpbind command binds CPU resources. - disable The pjpbind command does not bind CPU resources.

Environment variable name	Description
	If this environment variable is omitted, the operation is the same as with "enable" specified.

C.5 Setting the Range of CPU Resources Available to a Job [PG]

In an environment where Intel(R) Hyper-Threading Technology is enabled for MPI programs of an MPI processing system other than Development Studio, the job ACL function sets the range of the CPU resources of compute nodes available to a job. "Only the logical CPUs for the job" or "all logical CPUs" are set as the CPU resource range.

Figure C.1 Range of CPU resources available to a job



The range of the CPU resources of compute nodes available to a job is set by the job ACL function. The range can also be set by a job script. To use a job script to set the CPU resource range, use the pjpbind command, which is described in "C.2 Binding CPU Resources to Processes [PG]." In the job script, for reference by the pjpbind command, specify the following environment variable names and their setting values according to the CPU resource range to be set.

Table C.3 Environment variable names and setting values referenced by the pjpbind command when setting the CPU resource range

CPU resource range to set	Environment variable name and setting value
Only the logical CPUs for the job	PLE_ASSIGN_LOGICAL_CPU=job
All logical CPUs	PLE_ASSIGN_LOGICAL_CPU=all

If the environment variable PLE_ASSIGN_LOGICAL_CPU is not set, or if the specified value is neither "job" nor "all," the range of CPU resources available to a job conforms to the relevant setting of the job ACL function.

The following examples of job scripts set the range of CPU resources available to a job.

[Example1] Job script executing an MPI program (a.out) with "only the logical CPUs for the job"

```
#!/bin/sh
#PJM -L "node=4"
#PJM --mpi "proc=16"
...
export PLE_ASSIGN_LOGICAL_CPU=job
mpiexec.hydra pjpbind a.out
```

By using a job script to set the range of CPU resources available to a job, you can set the CPU resource range for each MPI program.

[Example2] Job script executing an MPI program (a.out) with "only the logical CPUs for the job" and another MPI program (b.out) with "all logical CPUs"

```
#!/bin/sh
#PJM -L "node=4"
#PJM --mpi "proc=16"
...
export PLE_ASSIGN_LOGICAL_CPU=job
mpiexec.hydra pjpbind a.out
```

```
export PLE_ASSIGN_LOGICAL_CPU=all
mpiexec.hydra pjpbind b.out
```

Information

If there are three or more logical CPUs, set one of them as the logical CPU for the system and all the rest as the logical CPUs for jobs.

Appendix D Operations on Jobs

This appendix shows whether operations can be performed on jobs depending on the job state and job type.

The meanings of the symbols in the tables below are as follows.

```
O: Operation on the job is permitted.
X: Operation on the job is not permitted.
-: The job never enters that status.
```

The meanings of the cells that are split in two under the Bulk job or Step job column are as follows.

```
Top: Operation on the bulk job or step job (job ID specified as the operation target)
Bottom: Operation on a sub job of the bulk job or step job (sub job ID specified as the operation
target)
```

The cells that are not split in two refer commonly to both to the bulk jobs and step jobs and their sub jobs.

Table D.1 Deleting a job (pjdel)

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
ACCEPT	O	O	O	O
REJECT	X	X	X	X
QUEUED	O	O	O	O
RUNNING-A	O	-	-	O
		O	O	
RUNNING-P	X(*)	-	X(*)	X(*)
		X(*)		
RUNNING	O	O	O	O
RUNNING-E	X(*)	-	X(*)	X(*)
		X(*)		
RUNOUT	X	O	-	O
		X	X	
CANCEL	X	X	X	X
ERROR	O	O	O	-
EXIT	X	X	X	X
HOLD	O	O	O	-
SUSPEND	O	O	O	-
SUSPENDED	O	O	O	-
RESUME	O	O	O	-

(*) If you specify the --enforce option, the operation is possible. However, you need to have the privileges to specify the --enforce option.

Table D.2 Holding a job (pjhold)

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
ACCEPT	X	X	X	X
REJECT	X	X	X	X
QUEUED	O	O	O	X

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
RUNNING-A	O(*1)	-	-	X
		O(*1)	O(*1)	
RUNNING-P	X(*2)	-	X(*2)	X
		X(*2)		
RUNNING	O(*1)	O(*3)	O(*1)	X
		O(*1)		
RUNNING-E	X(*2)	-	X(*2)	X
		X(*2)		
RUNOUT	X	X	-	X
			X	
CANCEL	X	X	X	X
ERROR	X	X	X	-
EXIT	X	X	X	X
HOLD	X	X	X	-
SUSPEND	O	O	O	-
SUSPENDED	O	O	O	-
RESUME	O	O	O	-

(*1) If you specify the --norestart option in the pjsub command when submitting the job or sub job, the operation is not possible.
(*2) If you specify the --enforce option, the operation is possible. However, you need to have the privileges to specify the --enforce option.
(*3) The target is sub jobs that can be held in bulk jobs. Sub jobs that cannot be held are ignored.

Table D.3 Release a held job (pjrls)

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
ACCEPT	X	X	X	X
REJECT	X	X	X	X
QUEUED	X	X	X	X
		X		
RUNNING-A	X	-	-	X
		X	X	
RUNNING-P	X	-	X	X
		X		
RUNNING	X	O(*)	X	X
		X		
RUNNING-E	X	-	X	X
		X		
RUNOUT	X	O(*)	X	X
		X		
CANCEL	X	X	X	X
ERROR	X	X	X	-

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
EXIT	X	X	X	X
HOLD	O	O	O	-
SUSPEND	X	X	X	-
SUSPENDED	X	X	X	-
RESUME	X	X	X	-

(*)The target is sub jobs that can be released the holding status in bulk jobs. Sub jobs that cannot be released the holding status are ignored.

Table D.4 Waiting a job (pjwait)

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
ACCEPT	O	O	O	O(*)
REJECT	O	O	O	O(*)
QUEUED	O	O	O	O(*)
RUNNING-A	O	-	-	O(*)
		O	O	
RUNNING-P	O	-	O	O(*)
		O		
RUNNING	O	O	O	O(*)
RUNNING-E	O	-	O	O(*)
		O		
RUNOUT	O	O	-	O(*)
			O	
CANCEL	O	O	O	O(*)
ERROR	O	O	O	-
EXIT	O	O	O	O(*)
HOLD	O	O	O	-
SUSPEND	O	O	O	-
SUSPENDED	O	O	O	-
RESUME	O	O	O	-

(*) The pjwait command ends normally without waiting for an interactive job to end or displaying any information.

Table D.5 Sending a signal to a job (pjsig)

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
ACCEPT	X	X	X	X
REJECT	X	X	X	X
QUEUED	X	X	X	X
RUNNING-A	X	-	-	X
		X	X	
RUNNING-P	X	-	X	X

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
		X		
RUNNING	O	O	O	O
RUNNING-E	X	- X	X	X
RUNOUT	X	X	- X	X
CANCEL	X	X	X	X
ERROR	X	X	X	-
EXIT	X	X	X	X
HOLD	X	X	X	-
SUSPEND	X	X	X	-
SUSPENDED	X	X	X	-
RESUME	X	X	X	-

Table D.6 Changing job parameter (pjalter)

Job status	Normal job, Master-worker job	Bulk job	Step job	Interactive job
ACCEPT	X	X	X	X
REJECT	X	X	X	X
QUEUED	O	O X	O(*1)	X
RUNNING-A	X	- X	- X	X
RUNNING-P	X	- X	X	X
RUNNING	O (*2)	O (*2)	O (*2)	X
RUNNING-E	X	- X	X	X
RUNOUT	X	X	- X	X
CANCEL	X	X	X	X
ERROR	O	O X	O(*1)	-
EXIT	X	X	X	X
HOLD	O	O X	O(*1)	-
SUSPEND	X	X	X	-
SUSPENDED	X	X	X	-
RESUME	X	X	X	-

(*1) The resource unit name cannot be changed for any operation on a sub job.
(*2) Only the elapsed time limit value for a job running on an FX server can be changed (when permitted by the administrator in the job operation management function settings). However, a change made with end-user privileges can only shorten the elapsed time limit value.

Appendix E Using Job Execution Environment

E.1 Creating an Image File for a Job Execution Environment

You need to create an image file to prepare a job execution environment yourself. This appendix has procedures for each job execution environment.

E.1.1 In Docker mode

You can use any method to create the image as long as you observe the precautions shown in "2.3.8 Specifying a job execution environment."

If you want to use the created container image as a job execution environment, check with the administrator about available execution methods (SDI specification and UDI specification). For SDI specification, request the administrator to register the container image in the system.

Information

The following packages and settings for using FX server-specific features are required to operate the Job Operation Software functions in the FX server container.

[Package List]

In the container image, install the following packages containing the Job Operation Software (See "Table E.1 List of packages required to operate the Job Operation Software functions) and the OS packages required by these packages (See "Table E.2 List of OS packages required to operate the Job Operation Software functions"). It is recommended to use the same number of versions of the OS as the compute nodes that have been proven to run the Job Operating Software functions. Contact your administrator to obtain the package for the job operation software.

In order to prevent the container image from becoming too large, the OS is based on installing only the Core group packages of Red Hat Enterprise Linux 8, and only the minimum required additional packages are listed here. If you are not satisfied with the functionality of the container, add packages accordingly.

Table E.1 List of packages required to operate the Job Operation Software functions

FJSVpxkrm
FJSVpxkrm-uti
FJSVpxpsm
FJSVpxpwrm_api
FJSVpxtof
FJSVxosfhehpc
FJSVxoshpcpwr
FJSVxoshwb
FJSVxoslibmpg
FJSVxoslibmpg-module
FJSVxosmemutils
FJSVxossec
docker-ce
libpfm
libpfm-devel
papi
papi-devel

papi-libs
xpmem
xpmem-kmod

Table E.2 List of OS packages required to operate the Job Operation Software functions

coreutils
elfutils-devel
elfutils-libelf-devel
gcc
gcc-c++
libstdc++-devel
libatomic
libevent-2
openssl-devel
zlib-devel

[Required settings]

The administrator must set the directory of the host environment (mount point), which is required for the job operation software to run when the container starts, in the startup configuration file in advance. Ask your administrator for this setting.

E.1.2 In McKernel mode

Obtain the source code from the public site (https://github.com/ihkmckernel). Follow instructions in README located in the source tree to build a McKernel image file (mckernel.img).

If you want to use the created image as a job execution environment, check with the administrator about available execution methods (SDI specification and UDI specification). For SDI specification, request the administrator to register the image in the system.

E.1.3 In KVM mode

A user on an active workstation creates a virtual machine image by using the virt-install utility and the virsh command-line interface.

If you want to use the created image as a job execution environment, check with the administrator about available execution methods (SDI specification and UDI specification). For SDI specification, request the administrator to register the image in the system.

[Virtual machine requirements]

Virtual machine image that is created with the virt-install utility and the virsh command-line interface is described below.

To create a virtual machine image file, use the virt-install utility. The domain name specified by virt-install (--domain) or the path to the virtual machine image (--disk) is optional. The domain XML file (*<Domain Name>*.xml) that is created at the same time that the virtual machine image file is created is required to start the virtual machine using the virsh command, but is not used by job operation software.
The virtual machine image file has the same format as that supported by QEMU.
The virtual machine must have a Linux kernel installed that supports virtiofs (Red Hat Enterprise Linux 8.2 or higher recommended).

See

For more information about the virtual machine, see "PART IX. VIRTUALIZATION ON ARM 64 SYSTEMS" in the Red Hat document "Red Hat Enterprise Linux 8 System Design Guide". Contact administrator to obtain this document.

In addition, the following requirements must be met when using KVM mode with the Job Operation Software.

- SELinux must be disabled.

- The sshd package is installed.
- Network settings are configured so that IP addresses can be assigned via DHCP.
- There must be a connection that is not associated with the device name.

  To meet this requirement, do the following within the virtual machine:

  - Adding a connection that is not tied to device names (Example with connection name "kvm")

    ```
    # nmcli connection add type ethernet ifname '*' con-name kvm
    Connection 'kvm' (cd871c4d-01ce-4335-8dbb-53d064d9d9ce) successfully added.
    ```

  - Checking a connection definition

    When a connection is added, ifcfg-*<connection name>* (In this example, 'ifcfg-kvm') under /etc/sysconfig/network-scripts/.
    The points to check are as follows.

    - 'BOOTPROTO = dhcp'.
    - Must be 'ONBOOT = yes'.
    - The line with 'DEVICE =' does not exist (Not tied to device name).

      ```
      # cat /etc/sysconfig/network-scripts/ifcfg-kvm
      TYPE=Ethernet
      PROXY_METHOD=none
      BROWSER_ONLY=no
      BOOTPROTO=dhcp
      DEFROUTE=yes
      IPV4_FAILURE_FATAL=no
      IPV6INIT=yes
      IPV6_AUTOCONF=yes
      IPV6_DEFROUTE=yes
      IPV6_FAILURE_FATAL=no
      IPV6_ADDR_GEN_MODE=stable-privacy
      NAME=kvm
      UUID=cd871c4d-01ce-4335-8dbb-53d064d9d9ce
      ONBOOT=yes
      ```

  - (Information) How to delete a connection.

    ```
    # nmcli connection del kvm
    Connection 'kvm' (cd871c4d-01ce-4335-8dbb-53d064d9d9ce) successfully deleted.
    ```

- The resource management agent package of the Job Operation Software is installed.

Note

The resource management agent package of the Job Operation Software is designed to operate with the init system services of systemd. To uniquely operate the resource management agent with an init system other than systemd, use the source archive of the agent to build a system appropriate to your virtual machine environment. Then, deploy the agent. For the build procedure, see the README supplied with the source archive.
The resource management agent package (FJSVpxkrm-libvirt-agent) and the source archive of the agent (FJSVpxkrm-libvirt-agent-*<version>*.src.tar.gz) are included in the Technical Computing Suite DVD. To procure them, contact the administrator.

Information

You can use the virsh command-line interface to start and stop virtual machines. To confirm in advance that a virtual machine image works, confirm that the resource management agent (pxkrm-libvirt-agent.service) in the virtual machine also starts normally.
The following is an example for checking the operation.

```
# virsh define <Path to the domain_XML_file(DomainName.xml)> (example: /etc/libvirt/qemu/test-
domain.xml)
# virsh start --console <DomainName> (example: test-domain)
(From here, virtual machine is being accessed)
login:
password:
...
# systemctl status pxkrm-libvirt-agent.service    (Confirming with the systemctl command)
* pxkrm-libvirt-agent.service - Technical Computing Suite - Kernel Resource Manager: libvirt agent
   Loaded: loaded (/usr/lib/systemd/system/pxkrm-libvirt-agent.service; enabled; vendor preset:
disabled)
   Active: active (exited) since Wed 2019-12-04 14:26:53 JST; 6min ago
  Process: 1490 ExecStart=/usr/sbin/pxkrm-libvirt-agent_ctl start (code=exited, status=0/SUCCESS)
 Main PID: 1490 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/pxkrm-libvirt-agent.service
           mq1497 /usr/libexec/FJSVtcs/krm/libvirtagent

Dec 04 14:26:54 tcs-base-vm.fujitsu.com libvirtagent[1497]: [INFO] [krm] 9999...
Dec 04 14:26:54 tcs-base-vm.fujitsu.com libvirtagent[1497]: [INFO] [krm] 9999...
...

or
# ps -ef | grep krm | grep libvirt      (Confirming with the ps command)
root      3178     1  0 Sep24 ?        00:33:05 /usr/libexec/FJSVtcs/krm/libvirtagent
root      3178     1  0 Sep24 ?        00:33:05 /usr/libexec/FJSVtcs/krm/libvirtexec
...
# shutdown -h now
...
(Up to here, virtual machine is being accessed)
# virsh undefine <DomainName>
```

E.2 Troubleshooting

This section describes what to do if a job submitted with a specified job execution environment fails to complete successfully. If the problem persists, contact your administrator.

E.2.1 Job ended with PJM code 28

If the PJM code (Exit Code) for the job is 28, it could be because:

- Wrong specification of job execution environment.

  The name of the job execution environment specified by the -L jobenv option of the pjsub command may be incorrect.

  ```
  $ pjsub -L jobenv=container job.sh
  (*) Job exection environment name 'container' is incorrect.
  ```

  If the job execution environment specified at the time of job submission is incorrect, specify the correct job execution environment.

If the specified job execution environment is correct, contact your administrator for the following problems:

- Container image or virtual machine image configuration problem. (Docker mode (SDI), KVM mode (SDI))
  You may have set up the container image in Docker mode or the virtual machine image in KVM mode incorrectly.
- Problems with the container image or the virtual machine image itself. (Docker mode (SDI), KVM mode (SDI))
  Container images in Docker mode or virtual machine images in KVM mode may be incorrect.
- Out of memory allocated to the job or McKernel image problem. (McKernel mode (SDI))

E.2.2 Job ended with PJM code 29

If the PJM code (Exit Code) for the job is 29, it could be because:

- Container image problems (Docker mode (UDI))

  Check if the container image specified by the -x option of pjsub command in the environment variable PJM_JOBENV_DOCKER_IMAGE is in the wrong path or in the wrong file format.

```
$ pjsub -L jobenv=custom-docker -x PJM_JOBENV_DOCKER_IMAGE=/directory/my-docker.tar job.sh
(*)Container image path/directory/my-docker.tar is wrong or malformed.
```

  If you are having problems with the file format, use the docker export command to archive the specified container image in tar format.

  If you are satisfied with the path and file format of the container image, the container image may have failed to boot the container. Modify the specified container image appropriately.

- McKernel image problems (McKernel mode (UDI))

  Verify that the -x option of the pjsub command does not specify the PJM_JOBENV_MCKERNEL_IMAGE environment variable, or that the path of the McKernel image specified by the PJM_JOBENV_MCKERNEL_IMAGE environment variable is correct.

```
$ pjsub -L jobenv=custom-mckernel job.sh
(*) The environment variable PJM_JOBENV_MCKERNEL_IMAGE is not specified.

$ pjsub -L jobenv=custom-mckernel -x PJM_JOBENV_MCKERNEL_IMAGE=/directory/my-mck.img job.sh
(*)The McKernel image path specified by the environment variable PJM_JOBENV_MCKERNEL_IMAGE
   is incorrect.
```

  If the way you specified the McKernel image is correct, the McKernel image itself may be incorrect. Modify the McKernel image appropriately based on information from the McKernel public site.

- Virtual machine image problems (KVM mode (UDI))

  Verify that the -x option of the pjsub command does not specify the PJM_JOBENV_KVM_IMAGE environment variable, or that the path of the virtual machine image specified by the PJM_JOBENV_KVM_IMAGE environment variable is correct.

```
$ pjsub -L jobenv=custom-kvm job.sh
(*) The environment variable PJM_JOBENV_KVM_IMAGE is not specified.

$ pjsub -L jobenv=custom-kvm -x PJM_JOBENV_KVM_IMAGE=/directory/my-kvm.img job.sh
(*)The virtual machine image path specified by the environment variable PJM_JOBENV_KVM_IMAGE
   is incorrect.
```

  If the way you specified the virtual machine image is correct, the virtual machine image itself may be incorrect and QEMU may fail to boot the virtual machine via libvirt. QEMU may have failed to start a virtual machine via libvirt. Modify the virtual machine image appropriately according to its requirements. It is recommended that you verify that the virtual machine is up and running beforehand.

E.2.3 Job ended with PJM code 140

If the PJM code (Exit Code) for the job is 140, it could be because:

- Multiple KVM jobs have been assigned to the same node. (KVM Mode)

  Check if the job is a node-exclusive job allocated to one node.

```
$ pjsub -L vnode=1,jobenv=custom-kvm -x PJM_JOBENV_KVM_IMAGE=/directory/my-kvm.img job.sh
(*) The job is submitted as a node-sharing job (vnode=1) to which virtual nodes are assigned.
```

  To run as a node-exclusive job, assign a physical node(-L node=1). For PRIMERGY servers, assigning a virtual node in SIMPLEX mode (-L vnode=1 -P exec-policy=simplex) is also a node-exclusive job.