Chapter 15

High Performance Big Data Research Team

15.1 Members

Kento Sato (Team Leader)

Jens Domke (Postdoctoral Researcher)

Jian Guo (Postdoctoral Researcher)

Takaaki Fukai (Postdoctoral Researcher)

Tonmoy Dey (Intern)

Rupak Roy (Intern)

Toshio Endo (Visiting Scientist)

Akihiro Nomura (Visiting Scientist)

Takashi Shimokawabe (Visiting Scientist)

15.2 Overview of Research Activities

The High Performance Big Data Research Team has been studying and developing software for accelerating "machine learning, deep learning and large-scale big data processing (AI techniques)" on the K and the Fugaku supercomputers (HPC for AI). We also study for accelerating HPC applications and HPC systems by using these AI techniques (AI for HPC). In FY2019, our team made several achievements which include: (1) Improving Data Compression with Deep Predictive Neural Network for Time Evolutional Data; (2) Breakdown of Modern HPC Applications: Less Double-Precision Floating-Point Units and Faster Memory are Needed; (3) The First Supercomputer with HyperX Topology: A Viable Alternative to State-of-the-Art Fat-Trees Topologies; (4) Counter-based Performance Extrapolation Toolchain; (5) Optimizing Asynchronous Multi-level Checkpoint/Restart Configurations with Machine Learning; (6) Evaluating the Relationship between System Utilization and Benefits of Combining Different Scheduling Policies with Backfilling.

15.3 Research Results and Achievements

15.3.1 Improving Data Compression with Deep Predictive Neural Network for Time Evolutional Data

A lot of intermediate data has to be generated and transferred for further analysis in data science. A Large Hadron Collider (LHC) in CERN generated about 88PB of data in 2018 and foresees "Data archival is expected to be two-times higher during Run 3 and five-times higher or more during Run 4 (foreseen for 2026 to 2029).".

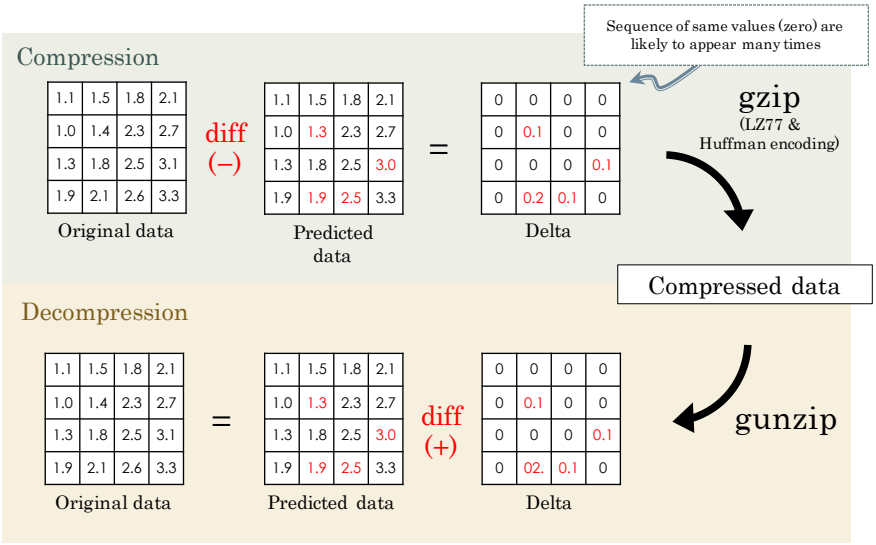

Figure 15.1: Predictive delta compression

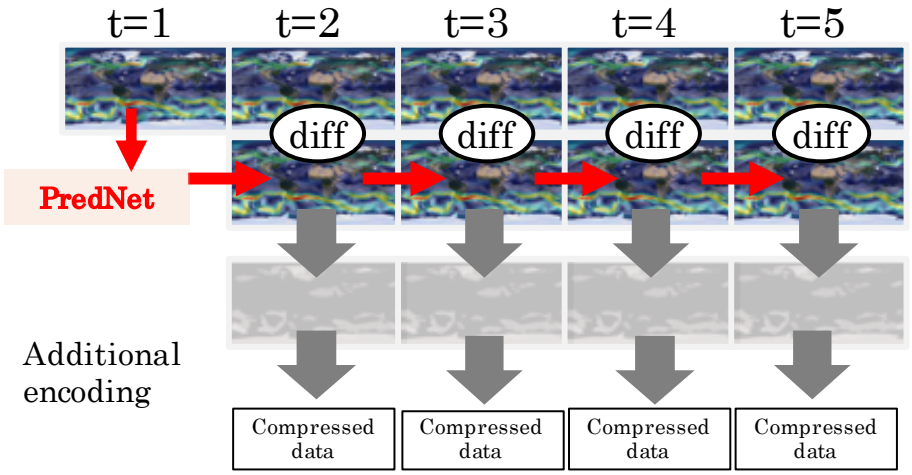

Figure 15.2: Data compression with predictive neural network

RIKEN also has a large synchrotron radiation facility (SPring-8). Spring-8 public beamline generated 0.32 PB/year in 2017. In 2025, with the next generation detector (CITISU), it is projected that a single beamline will generate 1.3 Exabytes of data per year. For the data analysis, checkpointing, debugging, visualization, etc., data generated by the scientific applications or simulations must be transferred from the sensors to computer systems. Fast transfer of such huge scale data from the sensors to computer systems is critical.

One of the approaches to accelerate data transfer is to reduce data size, i.e., data compression. However, existing compression algorithms show a low compression ratio for such kind of random evenly distributed floating-point data. As a result, achieving significant I/O acceleration is not possible with existing compressors. One of the promising approaches is predictive delta compression. Predictive delta compression is a technique to store only difference between original data and predicted data or the difference between consecutive predicted frames. Therefore, accurate prediction to the original data, which is data to compress, is important (Figure 15.1), both for increasing the compression ratio by making the delta values very small. Because the image data from sensors are time-evolutional images, we need a technique to accuracy predict future image frames.

Predictive neural network is a predictive coding based deep convolutional neural network which learns to predict future frames of a video sequence. PredNet is such an architecture which is trained to predict the future movement of objects. We use PredNet to predict the future frames from the given time evolutional frames. First, we train PredNet to learn movement of pixels by giving a number of time-evolutional frames generated from the sensor. In the example, in Figure 15.2, when we compress frames from t=2 to t=5, we predict future

frames from original frame (t=1), we compute the difference and then apply conventional compressors such as gzip. Since we can always generate the predicted frames with the original frame at t=1 with the help of trained neural network. we can restore the original frames by only storing (1) the original frame; (2) the trained neural network; (3) compressed frames from t=2 and t=5. For compressing the frames we pass the delta data through two steps. The steps are spatial delta encoding and entropy coding.

In the evaluation, we observe that we can compress SPring-8 data by a factor of 40 compared to original size. Our approach shows 2.8x time better compression ratio compared to a recent compression algorithm SZ.

15.3.2 Breakdown of Modern HPC Applications: Less Double-Precision Floating-Point Units and Faster Memory are Needed

Historically, most of the compute silicon has been allocated to double-precision (DP; 64-bit) compute. Nowadays, in processors such as AA64FX and NVIDIA Volta, the trend (mostly driven by market/AI demands) is to replace some of the double-precision units with lower-precision units. Lower-precision units occupy less area (up to 3x going from double- to single-precision FMA), leading to more on-chip resources (more instruction-level parallelism), potentially lowered energy consumption, and a definitive decrease in external memory bandwidth pressure (i.e., more values per unit bandwidth). The gains: up to four times over their DP variants with little loss in accuracy are attractive and clear, but what is the impact on performance (if any) on existing HPC applications? What performance impact can HPC users expect when migrating their code to future processors with a different distribution in floating-point precision support? Finally, how can we empirically quantify this impact on performance using existing processors in an apples-to-apples comparison on real-life use cases without relying on tedious, slow, and potentially inaccurate simulators?

To answer these questions, we selected two processors with identical micro-architecture, where the main difference is in the relative allocation of double-precision units, namely Intel's Knights Landing and Knights Mill architecture and a reference compute node which is commonly found in modern HPC systems. We stressed both processors with 22 HPC benchmarks and procurement application from the Exascale Computing Project and RIKEN's R-CCS Fiber Miniapp Suite. An in depth analysis of various aspects for these applications revealed that most of them are either memory-bound or perform little FP64 operations, as can be seen in the roofline plot in Figure 15.3.

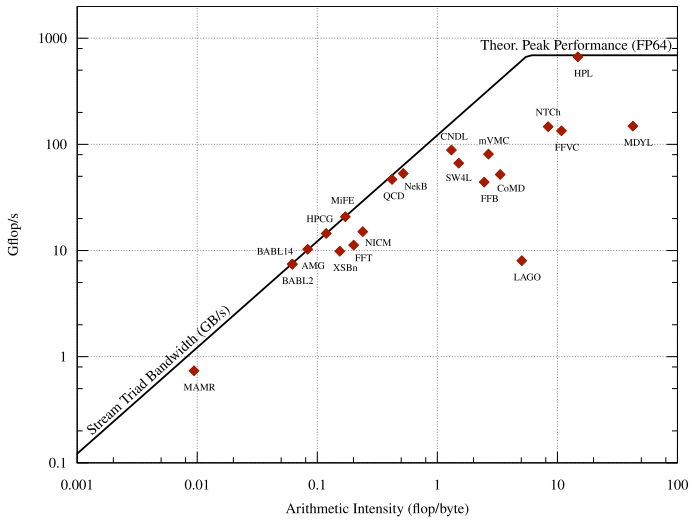

Figure 15.3: Roofline plot (w.r.t dominant floating-point operations and DRAM bandwidth) for Broadwell-EP reference system; Intel KNL/KNM reults omitted to improve visual appearance (they showed similar behavior)

By studying a large number of HPC proxy application, we found no significant performance difference between these two processors, despite one having more double-precision compute than the other. Our study points toward a growing need to re-iterate and re-think architecture design decisions in high-performance computing, especially with respect to precision. Our versatile evaluation toolchain which we developed for this study has been publicly released, alongside the peer-reviewed publication [2], and will allow follow-up and replication studies by researchers and industry.

15.3.3 The First Supercomputer with HyperX Topology: A Viable Alternative to State-of-the-Art Fat-Trees Topologies

The recent generation of supercomputers underwent a drastic scale-out effect, e.g., reaching the extremes of K or Sunway TaihuLight with over 80,000 and over 40,000 nodes, respectively, to tackle the ending of Moore's law. The interconnect in these HPC systems faces increasing demands for ultra-low latency from legacy HPC workloads, and new needs for high throughput of large messages from deep learning frameworks. Deploying Clos or Fat-Tree topologies will provide the needed throughput, but at a high cost. Furthermore, additional tree levels (to achieve this scale-out) will negatively affect the observable latency. Low-diameter, "electrically-optimized" topologies have been proposed, such as Dragonfly, Dragonfly+, or Slimfly. Another alternative is the HyperX which can provide high-throughput and low network dimensionality for ultralow latency.

We built the first large-scale HyperX prototype from the remains of the decommissioned TSUBAME2 supercomputer in collaboration with the Tokyo Institute of Technology and Hewlett Packard Enterprise. In a multi-months effort, we constructed the largest possible HPC system, given our hardware constraints, with HyperX network resulting in a 12x8 2D topology with 7 nodes per switch. The 672 compute nodes and 96 InfiniBand edge switches, composing our HyperX network, were distributed over 24 compute racks. The resulting system with two separate network rails, one of the old fat-trees and the new HyperX, see Figure 15.4, allowed for a near-perfect 1-to-1 comparison between the two topologies, which we published in two peer-reviewed conferences [3] and [4]. For these studies, we not only executed numerous MPI benchmarks and parallel HPC applications, but also developed a novel routing algorithm (PARX) to overcome the missing adaptive routing capabilities in the InfiniBand hardware.

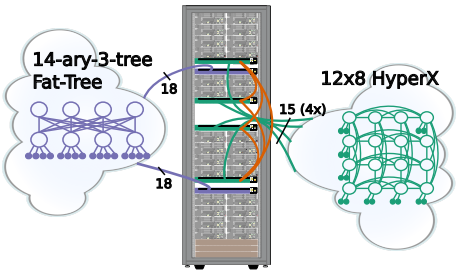

Figure 15.4: Depiction of one of 24 racks of our 672-node supercomputer with two edge switches connecting to the 3-level Fat-Tree and four switches for the 12x8 HyperX network (brown = rack-internal passive copper cables; gaps used for cable management)

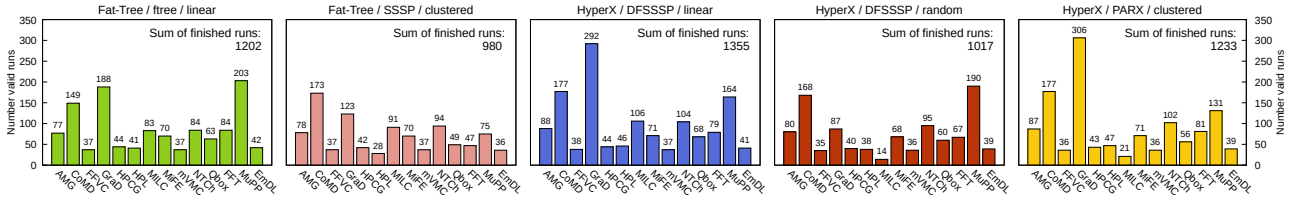

Figure 15.5: Capacity run for all five combinations over a 3 h time period for 14 concurrently running applications (using 32 or 56 nodes) while occupying 98.8% of the supercomputer; "HyperX / DFSSSP / linear" yields the highest number of finished compute jobs, outperforming the Fat-Tree by 12.7%, followed by PARX routing which yields 3% higher throughput.

The collected data from our 1-to-1 comparison implies that even a HyperX topology with roughly half-bisection bandwidth, and hence drastically reduced deployment costs, can compete with our 18-ary 3-tree, which theoretically offers more than full-bisection due to the reduced node count at the leafs. This result is even more astonishing, considering that we only had deprecated IB equipment (QDR type) available, which does not feature the required adaptive routing for the HyperX topology. We investigated two strategies: MPI rank placement and our novel PARX routing, which shows great potential as depicted in Figure 15.5, to circumvent the bottleneck arising from applying a shortest-path, static routing to a HyperX.

15.3.4 Counter-based Performance Extrapolation Toolchain

Nowadays, co-design efforts are aided by simulators, such as RIKEN's gem5-based architecture simulator for Supercomputer Fugaku. However, such simulators have severe drawbacks. First, development cost (human time and labor) is substantial while still delivering runtime estimation errors in the lower double-digit percentage area. Second, this labor-intensity and required computer architecture knowledge means that a near-accurate simulation approach can only be used towards the end of a co-design effort, when the system deployment is only few years or months away. And lastly, simulator have reported slowdowns of 1000-10,000x compared to executing the program on real hardware, and therefore only small "toy-codes" or application hotspots can be tested, instead of full scientific programs running on future HPC systems.

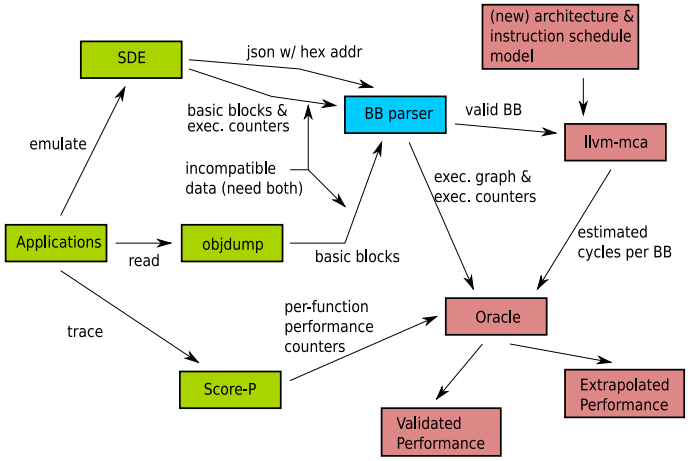

Figure 15.6: Counter-aided Performance Extrapolation Toolchain

We are currently exploring an alternative approach [8], by combining various tools into a framework designed to quickly test new ideas and extrapolate the performance of known/legacy application into the 5-to-10 year future. This toolchain, see Figure 15.6 shall aid processor and full-system architects in their early "what-if" stages to estimate the effect of a proposed architecture change for the full application runtime. Using existing hardware, we extract performance counters/data and internal information of the application, such as instruction- and memory-traces, and correlate this data with the basic blocks of the program. Defining a future node architecture and adjusting architecture characteristics, e.g., memory bandwidth and latency, can then be used to extrapolate the effects of such changes for individual basic blocks. Hence, knowing the speed up for each block yields an estimate of the performance benefit of the full application gained from a architecture.

15.3.5 Optimizing Asynchronous Multi-level Checkpoint/Restart Configurations with Machine Learning

To reliably run applications on current petascale systems in High-performance computing (HPC) , a commonly used technique is checkpoint/restart(CR). In CR, the system writes a snapshot of the application's state at fixed intervals to the different levels of storage hierarchy based on the configuration of the system. Though checkpoint and restart are useful for large scale systems, the checkpointing overhead can be enormous in an extensive system, which becomes one of the issues in the CR method.

One of approaches to reduce the overhead of CR is to determine the optimal checkpoint interval and checkpoint count. Poorly determined checkpoint interval makes system resilience worse. There are two approaches for obtaining the optimal checkpoint interval and checkpoint count values for any given configuration, namely the modeling approach and the simulation approach. The modeling approach mainly formulates an analytical solution to obtain optimal values, whereas the simulation approach runs the application across multiple failures to check different scenarios for obtaining the optimal values.

In this research, we try to use machine learning models in combination with an accurate simulation approach to determine the optimized checkpoint configuration. Specifically, the simulator has been developed to replicate the behavior of real-world scenarios when using a multi-level checkpoint for large scale systems. The simulator is provided with three critical parameters for each level, checkpoint overhead, check-point restart time, and failure rates as shown in Figure 15.7.

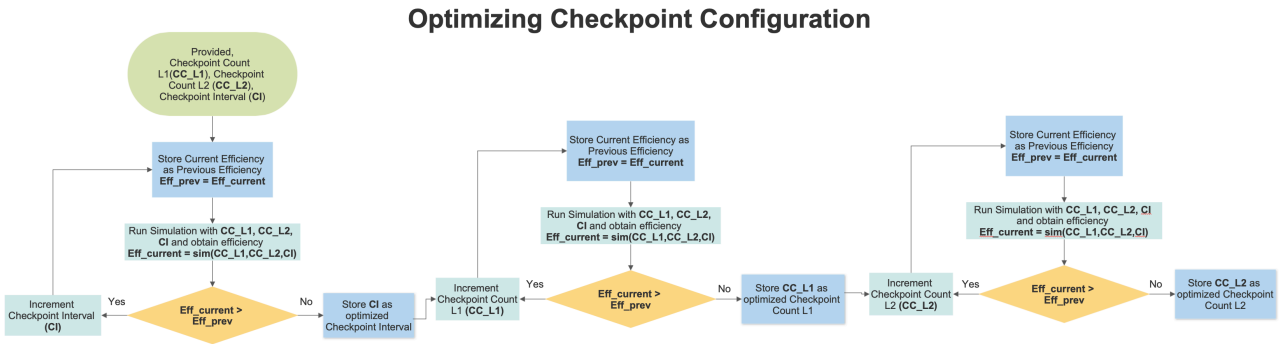

Figure 15.7: Workflow for Optimizing Checkpoint Configuration

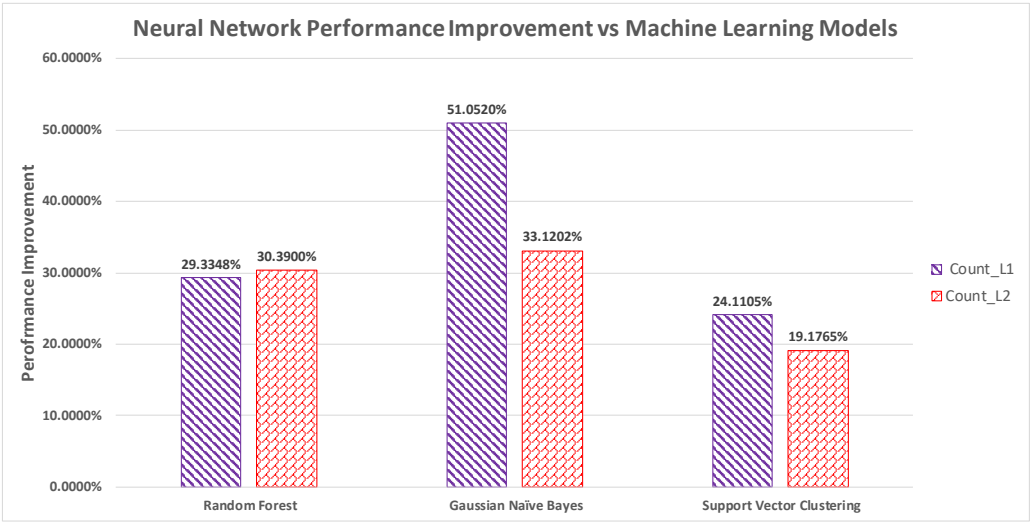

Figure 15.8: Neural Network Models Performance Improvement vs Machine Learning Models for three-level checkpoint model

The above simulation run provides the elapsed time and efficiency of a system with a specific configuration. The simulator we have developed not only provides the elapsed time and efficiency for a system but also performs simulation across different settings of the checkpoint system to determine the optimal checkpoint count for each level of the multi-level checkpoint system. The optimized configurations are obtained by modulating the checkpoint configuration values starting with checkpoint interval. After the peaks for all the configuration parameters are captured by the simulator, the user is provided with the optimized configuration for the given system based on the overhead, restart time, and failure rates. Once a significant amount of data is collected from the simulator, the information is passed on to different machine learning and neural network models to predict the optimized checkpoint configuration for other systems with different over-head, restart time, and failure rates.

we present an idea to combine the simulation approach with machine learning models to reduce the time taken to determine the optimized parameter values of check-point interval and checkpoint count for different configurations of CR. With our approach and design optimizations, we show that our models can predict the optimized parameter values when trained with the simulation approach. We have also demonstrated that using techniques such as neural networks can improve the performance over the machine learning models with neural network sometime exceeding the performance of a machine learning model by 50% as Figure 15.8 shows.

15.3.6 Evaluating the Relationship between System Utilization and Benefits of Combining Different Scheduling Policies with Backfilling

HPC users are required to estimate the job execution time and computing resource of their jobs before submitting, then the resource and job management systems (RJMSs) will submit jobs to HPC systems through

different scheduling policies for job execution. Related studies have shown that the EASY-Backfilling, also known as EASY-FCFS (EASY-First-Come-First-Served) is the most widely used scheduling policy in a lot of HPC systems due to its simple and robust implementation and increasing the overall utilization of the platform. On the other hand, because each of the HPC systems has its own scheduling policies, applications and users, which results in different system utilizations. Although with the same scheduling policy, there will be different benefits under different system utilizations of HPC systems. In this paper, we used real workload logs-based simulation to deep mining and analyze the relationship between system utilization and the benefits of combining different scheduling policy with backfilling for job scheduling in HPC systems.

In this research, we used real workload logs-based simulation to deep mining and analyze the relationship between system utilization and the benefits of different backfilling scheduling policies for job scheduling in HPC systems. Specifically, we would like to research the difference about the benefits of combining different scheduling policies with backfilling on different HPC systems. Meanwhile, we also want to explore and analyze how about the performance gap between different scheduling policies on low and medium utilization systems? And is a backfilling mechanism necessary on high utilization systems?

15.3.6.1 Current Progress

Currently, we have done a preliminary analysis on the three datasets (CEA Curie, LLNL Thunder and AIST AAIC): including the distribution of user estimated job execution time / real job execution time, the distribution of job states, etc. Firstly, we would like to check the real job execution time which was recorded after job execution to see whether the job execution time by a user conforms to a certain distribution law, e.g. a normal distribution? We analyse the distribution the real job execution time based-on the user-vector which is generated by combination of userID, groupID, queue, nodes, req_walltime from SWF file (A file format for saving HPC system log). The reason why we use user-vectors for splitting jobs because we assume that vectors with the same information (same userID, groupID, queue, nodes and req_walltime) represent that the user is submitting same applications for same purpose. Figure 15.9 is an example of distribution the real job execution time based-on the user-vector generated by using real workloads from CEA Curie dataset, we can see that the distribution the real job execution time of this user is random distribution which cannot be easily predicted.

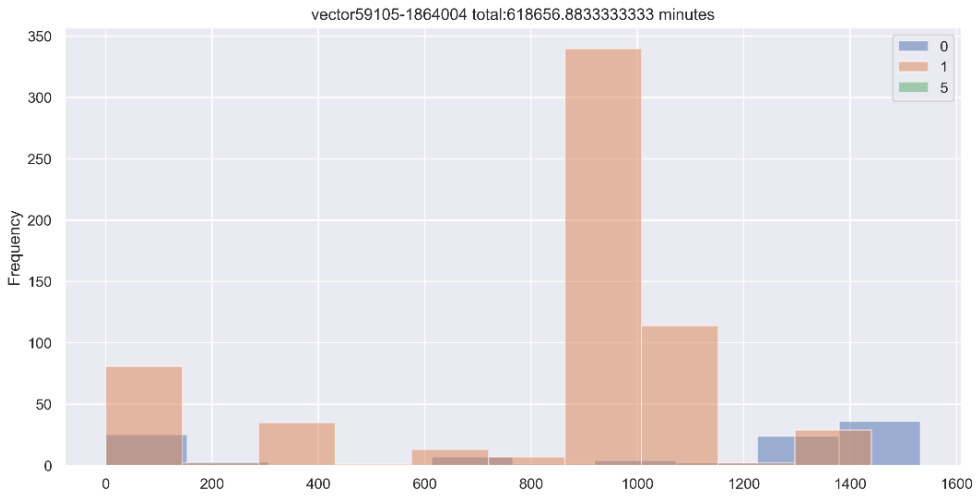

Figure 15.9: Histogram example of real job execution time from CEA Curie system, which is a user who submitted a total of 618,657 minutes.

Then, we also analyze the relationship between submit time and real job execution time base on user-vectors. We would like to see whether jobs submitted over a period of time by one user have the similar execution time. From Figure 15.10 we can see that job submitted over a period of time by one user has similar execution time. This phenomenon is common in other users. Generally speaking, job scheduling-related experiments cannot be performed directly on a real HPC system at the beginning. Most researchers choose to do their research with using job scheduling simulators. After deployment and adjustment of the simulators, we would like to start our

Experimental designing for our research. This research will base on a hypothesis that there is a relationship between different system utilization and the benefits of combining different scheduling policies with backfilling.

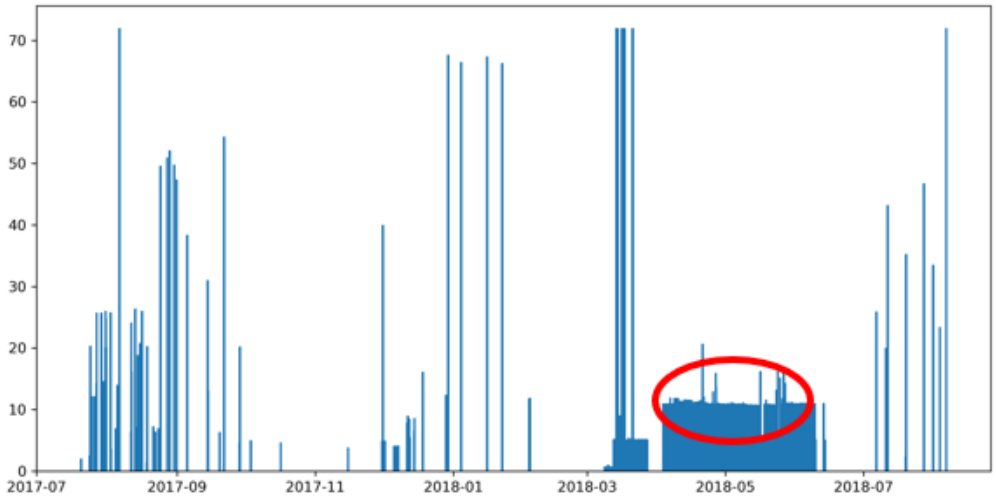

Figure 15.10: Jobs submitted over a period of time by one user have the similar execution time.

In this research, we also would like to conduct our experiments with simulation. After examining the popularity in recent research paper, updating frequency, functionality, and ease of use, we tentatively decided to use two simulators, BatSim and Slurm Simulator as our experiment platforms of job scheduling simulation in our research.

Future Plan

If everything goes well, we would like to follow tasks remain for the 2020 fiscal year:

- Extend this research to optimize job scheduling by predicting job execution time with AI
- Try to evaluate and test some of the latest AI-based (Machine Learning and Deep Learning-based) optimization studies for job scheduling in HPC systems

15.4 Schedule and Future Plan

In FY2020, we continuously work on HPC-for-AI, AI-for-HPC and many other researches and developments for HPC. These research topic include: (1) Fast and scalable parallel I/O by taking advantage of next-generation memory (e.g., Non-volatile memory) in big data processing and machine learning; (2) Scalable checkpointing for fault tolerance by taking advantage of next-generation memory (e.g., Non-volatile memory); (3) Scalable algorithms for deeply hieratical memory and storage architecture; (4) Fast data transfer technique for multi-petabyte of big data on high-speed network; (5) Integration of software stacks of big data, machine learning and HPC, and their optimization; (6) Visualization and UI techniques of big data; (7) Other research and software development related to big data, machine learning and I/O.

15.5 Publications

15.5.1 Articles/Journal

[1] Chapp, D., Rorabaugh, D., Sato, K., Ahn, D. H., and Taufer, M. (2019). A three-phase workflow for general and expressive representations of nondeterminism in HPC applications. The International Journal of High Performance Computing Applications, 33(6), 1175–1184.

15.5.2 Conference Papers

[2] J. Domke, K. Matsumura, M. Wahib, H. Zhang, K. Yashima, T. Tsuchikawa, Y. Tsuji, A. Podobas, S. Matsuoka, "Double-precision FPUs in High-Performance Computing: an Embarrassment of Riches?," in

Proceedings of the 33th IEEE International Parallel & Distributed Processing Symposium (IPDPS), (Rio de Janeiro, Brazil), IEEE Computer Society, May 2019.

[3] J. Domke, S. Matsuoka, I.R. Ivanov, Y. Tsushima, T. Yuki, A. Nomura, S. Miura, N. McDonald, D.L. Floyd, N. Dube, "The First Supercomputer with HyperX Topology: A Viable Alternative to Fat-Trees?," peer-reviewed short paper presented at the 2019 IEEE 26th Symposium on High-Performance Interconnects (HOTI 26), Aug. 2019.

[4] J. Domke, S. Matsuoka, I.R. Ivanov, Y. Tsushima, T. Yuki, A. Nomura, S. Miura, N. McDonald, D.L. Floyd, N. Dube, "HyperX Topology: First at-scale Implementation and Comparison to the Fat-Tree," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, (Piscataway, NJ, USA), IEEE Press, Nov. 2019.

[5] T. Dey, K. Sato, B. Nicolae, J. Guo, J. Domke, W. Yu, F. Cappello, K. Mohror, "Optimizing Asynchronous Multi-level Checkpoint/Restart Configurations with Machine Learning," accepted at the IEEE International Workshop on High-Performance Storage (co-located with 34th IEEE IPDPS), May 2020.

15.5.3 Posters

[6] T. Dey, K. Sato, J. Guo, B. Nicolae, J. Domke, W. Yu, F. Cappello, K. Mohror, "Optimizing Asynchronous Multi-Level Checkpoint/Restart Configurations with Machine Learning," Poster presented at the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, (Piscataway, NJ, USA), IEEE Press, Nov. 2019.

[7] R. Roy, K. Sato, J. Guo, J. Domke, W. Yu, T. Hatsui, Y. Joti, "Improving Data Compression with Deep Predictive Neural Network for Time Evolutional Data," Poster presented at the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, (Piscataway, NJ, USA), IEEE Press, Nov. 2019.

[8] J. Domke, K. Sato, M. Kondo, "Counter-based Performance Extrapolation Toolchain – How far can we look into the Future?," Poster presented at The 2nd R-CCS International Symposium (RCCS-IS2), Kobe, Japan, Jan. 2020.

[9] K. Sato, A. Kuroda, K. Minami, J. Domke, A. Drozd, M. Wahib, S. Kudo, T. Imamura, K. Kumahata, K. Nitadori, K. Ando, S. Matsuoka, "DL4Fugaku: Deep learning for Fugaku – Scalability Performance Extrapolation," Poster presented at The 2nd R-CCS International Symposium (RCCS-IS2), Kobe, Japan, Jan. 2020.

[10] J. Guo and K. Sato, "Research on reproducibility and universality in machine learning-based optimizations for prediction job runtime in HPC systems," Poster presented at The 2nd R-CCS International Symposium (RCCS-IS2), Kobe, Japan, Jan. 2020.

15.5.4 Invited Talks

[11] J. Domke "First At-Scale HyperX Implementation: A Compelling Alternative to Fat-Trees?" in High Performance Consortium for Advanced Scientific and Technical Computing (HP-CAST 32), June 2019.

[12] 佐藤 賢斗, "高性能ビッグデータ処理とポストムーア時代に向けたアプリケーション解析", In JACORN (JApan Consortium for the Reconfigurable-hardware Next generation), October, 2019.

[13] Kento Sato, "Convergence of AI/Big data and HPC", In 4th International Symposium on Research and Education of Computational Science (RECS), October, 2019.

[14] Kento Sato, "AI for HPC - Data Compression and System Software Optimization", In France-Japan-Germany trilateral workshop: Convergence of HPC and Data Science for Future Extreme Scale Intelligent Applications, November, 2019.

[15] Kento Sato, "High performance AI training on SVE/A64FX", In Arm HPC User Group (AHUG) at SC19, November, 2019.

[16] 佐藤 賢斗, "「富岳」AI 利用の展望", In 第 14 回 AI サービス研究会, December, 2019.

[17] J. Domke "Double-precision FPUs in High-Performance Computing: an Embarrassment of Riches?" in Workshop on Large-scale Parallel Numerical Computing Technology (LSPANC 2020 January), Jan. 2020.

[18] Kento Sato, "Convergence of AI/BD and HPC - Data compression and DL4Fugaku", Inauguration Meeting of Synchrotron for Neuroscience – an Asia Pacific Strategic Enterprise (SYNAPSE), Singapore, January, 2020)

15.5.5 Oral Talks

[19] Kento Sato, "Model, Simulation and AI for Checkpointing", The 9th JLESC Workshop, April, 2019

[20] Kento Sato, "ANL/RIKEN Collaboration on Lossy Compression", The DOE/MEXT Meeting, May, 2019

[21] Kento Sato, "LLNL/UTK/RIKEN Collaboration — Data Analytics on System log —", The DOE/MEXT Meeting, May, 2019

[22] Franck Cappello, Bogdan Nicolae, Kathryn Mohror, Rupak Roy, Weikuan Yu and Kento Sato, "ANL/F-SU/RIKEN Collaboration On C/R Optimization with AI", The DOE/MEXT Meeting, May, 2019

[23] Kento Sato, "High Performance Big Data Research Team", The SPring-8 Meeting, August, 2019

[24] Kento Sato, "Data Compression with Deep Predictive Neural Network", BDEC2, October, 2019

[25] Kento Sato, Jens Domke, Jian GUO "Convergence of AI/Big data and HPC", R-CCS Cafe, November, 2019