PFAnalyzer: A Toolset for Analyzing Application-aware Dynamic Interconnects

Keichi Takahashi, Susumu Date, Dashdavaa Khureltulga, Yoshiyuki Kido, Shinji Shimojo

Cybermedia Center, Osaka University
Osaka, Japan
{takahashi.keichi, huchka}@ais.cmc.osaka-u.ac.jp, {date, kido, shimojo}@cmc.osaka-u.ac.jp

***Abstract*—Recent rapid scale out of high performance computing systems has rapidly and continuously increased the scale and complexity of the interconnects. As a result, current static and over-provisioned interconnects are becoming cost-ineffective. Against this background, we have been working on the integration of network programmability into the interconnect control, based on the idea that dynamically controlling the packet flow in the interconnect according to the communication pattern of applications can increase the utilization of interconnects and improve application performance. Interconnect simulators come in handy especially when investigating the performance characteristics of interconnects with different topologies and parameters. However, little effort has been put towards the simulation of packet flow in dynamically controlled interconnects, while simulators for static interconnects have been extensively researched and developed. To facilitate analysis on the performance characteristics of dynamic interconnects, we have developed PFAnalyzer. PFAnalyzer is a toolset composed of PFSim, an interconnect simulator specialized for dynamic interconnects, and PFProf, a profiler. PFSim allows interconnect researchers and designers to investigate congestion in the interconnect for an arbitrary cluster configuration and a set of communication patterns collected by PFProf. PFAnalyzer is used to demonstrate how dynamically controlling the interconnects can reduce congestion and potentially improve the performance of applications.**

***Index Terms*—Simulation, Profiling, Interconnect, Message Passing Interface, Software Defined Networking**

I. Introduction

Inter-node communication performance of High Performance Computing (HPC) clusters heavily affects the total performance of communication-intensive applications. Communication-intensive applications require low-latency and high-bandwidth communication between computing nodes to fully exploit the computational power and parallelism of the computing nodes. High performance networks that provide such low-latency and high-bandwidth communication between computing nodes of a cluster are often referred to as *interconnects*. Message Passing Interface (MPI) [1, 2] is a commonly used inter-process communication library to describe communication on HPC clusters.

In this paper, interconnects are roughly classified into *static interconnects* and *dynamic interconnects*. In the former category, it is assumed that packet flow is controlled solely based on its source and/or destination. A well-known exemplifier is InfiniBand [3], where forwarding tables of switches are populated with pre-computed forwarding rules in advance of the execution of applications. In contrast, in the latter category, it is assumed that packet flow is dynamically controlled to mitigate load imbalance and improve utilization of the interconnect.

Nowadays, the majority of HPC clusters employ the former static interconnects because of the low implementation cost. Since static interconnects are controlled without taking the communication patterns of individual applications into account, they are usually designed to be able to accommodate the worst-case traffic demand to achieve good performance for a variety of applications, each of which has a different communication pattern. Interconnect designers have respected criteria such as full bisection bandwidth and nonblocking networks.

The continuously growing demand for computing power from academia and industry has inevitably forced HPC clusters to scale out more and more. As a result of the growing number of computing nodes, interconnects are becoming increasingly large-scale and complex. This technical trend is making static and over-provisioned interconnects cost-ineffective and difficult to build.

Based on this background and these trends, we study the feasibility and applicability of programming dynamic interconnect within HPC [4]. In particular, *SDN-enhanced MPI* [5, 6], a framework that incorporates the dynamic network controllability of Software-Defined Networking (SDN) [7] into MPI, has been researched based on the idea that dynamically optimizing the packet flow in the interconnect according to the communication patterns of applications can increase the utilization of the interconnect and improve application performance. The goal of SDN-enhanced MPI is to accelerate individual MPI functions by dynamically optimizing the packet flow in the interconnect. Several MPI functions have been successfully accelerated in our previous works so far. One of the core challenges in the research of SDN-enhanced MPI is to develop algorithms to control the packet flow in the interconnect depending on the MPI function called by the application.

More generally, algorithms for efficiently controlling the packet flow in the interconnect depending on the communication patterns of applications is essential towards realizing a dynamic and application-aware interconnect. In order to develop a generic algorithm that achieves good performance on a variety of environments, the algorithm must be investigated and evaluated by targeting different applications and interconnects. However, utilizing physical clusters to analyze the performance characteristics of the interconnect is restricted in the following points. First, the execution time of real-world HPC applications typically ranges from hours up to days, sometimes even months. Second, large-scale deployments of dynamic interconnects that allow execution of highly parallel applications have not yet been seen because research and development of dynamic interconnects are still at their early stage. Third, network hardware such as switches may not support measuring traffic in the interconnect with enough high frequency and precision to obtain meaningful insights.

To accelerate the research and development of application-aware dynamic interconnects that control packet flow in response to the communication patterns of applications, an interconnect simulator that allows users to conduct systematic investigation of clusters with diverse topologies and parameters is vitally demanded. A wide spectrum of interconnect simulators have been developed with different focus and purpose until today. However, existing simulators mostly focused on static interconnects and little research has been done to simulate dynamic and application-aware interconnects.

This paper describes the design and implementation of PFAnalyzer, a toolset for analyzing application-aware dynamic interconnects. PFAnalyzer consists of two components: PFSim and PFProf. PFSim is an interconnect simulator specialized for application-aware dynamic interconnects. PFSim takes a set of communication patterns of applications and a cluster configuration as its input and then simulates the traffic on each link of the interconnect. PFProf is a custom profiler to extract communication patterns from applications for use in conjunction with PFSim.

The contributions of this paper are summarized as follows:

- A lightweight interconnect simulator for simulating dynamic and application-aware interconnects is proposed.
- A custom profiler for extracting communication patterns from applications is presented.
- Simulation results for NAS CG benchmark and MILC on a fat-tree interconnect are presented.

The rest of this paper is organized as follows. Section II examines the requirements of an interconnect simulator for dynamic and application-aware interconnects. Section III reviews related work. Section IV describes the design and implementation of PFAnalyzer. Section V presents the simulation results for NAS CG benchmark and MILC obtained with our proposed simulator. Furthermore, results of a verification experiment on a physical cluster are shown. Section
VI concludes this paper and outlines our future work.

II. Research Objective

In this paper, we aim to realize an interconnect simulator specialized for application-aware dynamic interconnects to facilitate the research and development of application-aware dynamic interconnects. Most interconnect simulators proposed in the previous research [8, 9, 10, 11] are designed to study the behavior of static interconnects. Therefore, packet flow is controlled based on static routing algorithms in these simulators.

However, packet flow needs to be dynamically controlled based on the communication patterns of applications for simulating application-aware dynamic interconnects. Based on this necessity, the requirements for our simulator are described as follows:

- *Support for application-aware dynamic routing*: The simulator should allow users to implement dynamic routing algorithms to mitigate load imbalance and improve utilization of the interconnect. In addition, the routing algorithms should be supplied with the communication patterns of applications and packet flow in the interconnect to make effective routing decisions.
- *Support for communication patterns of real-world applications*: Communication patterns of real-world HPC applications should be fed into the simulator to reproduce the characteristics of communication generated by real-world applications. To simulate the packet flow in the interconnect when an application is being executed, the simulator needs to predict the volume of point-to-point communication exchanged between computing nodes. In the actual computing scene, the traffic among computing nodes is generated by the processes executed on the computing nodes. Thus, some means and mechanisms to analyze the traffic volume of point-to-point communication exchanged between the processes from applications is essential. In this paper, applications that leverage MPI for inter-process communication are targeted.
- *Support for diverse cluster configurations*: The distribution of packet flow in the interconnect is determined by not only the routing algorithm but also the topology of the interconnect. In addition, job scheduling, node selection and process placement algorithms need to be considered. These parameters should be easily reconfigurable to allow users to perform a systematic investigation of diverse clusters.

Furthermore, the simulator should be designed to be lightweight and fast to carry out a large number of simulations with different parameters in a reasonable amount of time. If necessary, appropriate approximation should be introduced to improve simulation performance.

III. Related Work

Several interconnect simulators have been proposed in past research. PSINS [9] is a trace-driven simulator for HPC applications. Traces obtained from applications are used to predict the performance of applications on a variety of HPC clusters with different configurations. LogGOPSim [10] simulates the execution of MPI applications based on the LogGOP network model. Both simple synthetic communication patterns and communication patterns converted from traces of MPI applications can be fed into the simulator. A limitation of LogGOPSim is that the interconnect is assumed to have full bisection bandwidth and congestion is not simulated. These two simulators can provide accurate performance predictions owing to their per-message simulation capability. However, the topology and the routing algorithm of interconnects are abstracted in the network models of PSINS and LogGOPSim. Therefore, these simulators cannot be used for predicting and comparing the performance of different topologies or routing algorithms. In contrast, the simulator targeted in this paper allows users to compare the performance characteristic of different topologies and routing algorithms.

ORCS [8] simulates the traffic load of each link in the interconnect for a given topology, communication pattern and routing algorithm. The simulated traffic load of links can be summarized into various performance metrics and used for further analysis. A limitation of ORCS is that only pre-defined communication patterns can be used as its input. Moreover, ORCS assumes static routing as in InfiniBand. On the contrary, our simulator can handle dynamic routing algorithms that use communication patterns of applications and interconnect usage to make routing decisions.

In [11], simulations are carried out to examine the performance characteristics of an SDN-based multipath routing algorithm for data center networks. A simulator is developed based on MiniSSF to simulate the throughput and delay of a packet flow under diverse settings. However, communication patterns are randomly generated and not based on real-world applications. Our proposed simulator is designed to accept arbitrary communication patterns obtained from real-world applications using our custom profiler.

INAM^2 [12] is a comprehensive tool to monitor and analyze network activities in an InfiniBand network. The tight integration with the job scheduler and a co-designed MPI library allows *INAM*^{2} to associate network activities with jobs and MPI processes. For instance, it can identify hot spots in the interconnect and inspect which node, job, and process is causing the congestion. Although *INAM*^{2} is a useful tool for system administrators to diagnose the performance issues of interconnects, it is not suitable for studying diverse interconnects since it only supports physical clusters.

IV. Proposal

We propose *PFAnalyzer*, a toolset for analyzing the performance characteristics of application-aware dynamic interconnects. PFAnalyzer is composed of *PFProf*, a profiler to extract communication patterns from applications, and *PFSim*, a simulator capable of simulating application-aware dynamic interconnects.

A. Representation of a Communication Pattern

In this paper, we represent the communication pattern of an application using the traffic matrix of the application. For an application composed of n processes, its traffic matrix is defined as a $n \times n$ square matrix T of which element T_{ij} is equal to the volume of traffic sent from rank i to rank j. Here, we approximate the volume of traffic between processes as constant during the execution of a job. The traffic volume between a process pair is assumed to be the total bytes transferred divided by the duration of the application.

This approximation is introduced to simplify and speed up the simulation. The idea behind this approximation is based on the fact that many HPC applications (*e.g.* partial differential equation solvers) show an iterative nature. These applications spend most of their execution time inside a repetitive loop and thus their communication patterns do not significantly change over time. Therefore, the traffic matrix can be a good representation of the communication characteristics of an iterative application despite the loss of temporal order. In the future, we plan to apply trace segmentation [13] techniques on the communication trace. This technique will segment a trace into multiple communication phases, which could then be simulated individually to further improve the accuracy of simulation for applications with significantly time-varying communication patterns.

B. PFProf (MPI Profiler)

Initially, we tried to reuse existing MPI performance analysis tools such as Score-P [14], Vampir [15] and Tau [16] to collect the traffic matrices from MPI applications. However, these tools capture only a subset of the communication pattern when profiling an application that uses collective communication functions (*e.g.* MPI_Bcast, MPI_Allreduce and MPI_Reduce).

The reason can be explained as follows. Existing MPI profilers replace the standard MPI functions provided by MPI libraries with instrumented functions by utilizing the MPI Profiling Interface (PMPI). An advantage of this approach is that it works regardless of a specific MPI implementation. However, this approach fails to capture the internal information of the MPI library. Meanwhile, collective communication functions are internally implemented as a combination of point-to-point communication in MPI implementations. These underlying point-to-point communication functions are hidden from PMPI-based profilers and excluded from the communication patterns emitted by profilers.

To accurately capture the underlying point-to-point communication of collective communication, we have developed a profiler by utilizing the MPI Performance Examination and Revealing Unexposed State Extension (PERUSE) [17]. PERUSE was designed to provide internal information of the MPI implementation that were not exposed through PMPI to applications and performance analysis tools. By using PERUSE, application or performance analysis tools register callback functions for each event of interest. After that, the registered callback function is called each time when the associated event occurs.

Figure 1 illustrates how PFProf, MPI library and MPI application interact with each other. PFProf hooks MPI_Init and MPI_Finalize to perform initialization and finalization. During the initialization, PFProf subscribes to two PERUSE events: `PERUSE_COMM_REQ_XFER_BEGIN` and `PERUSE_COMM_REQ_XFER_END`. These events are emitted each time a transfer of a message begins and ends, respectively. Profiling results are written out as a JSON file during the finalization.

During the execution of an application, the total number of bytes transferred from a process to another is aggregated to compute the traffic matrix.

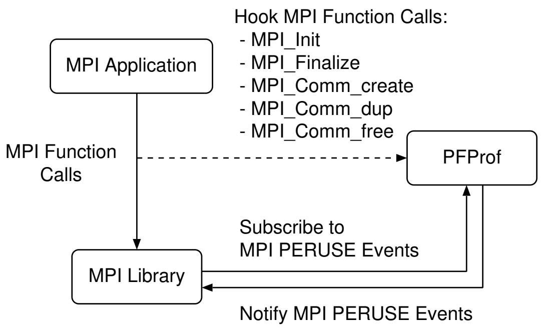


Figure 1: Block Diagram of PFProf

Furthermore, PFProf hooks several MPI functions to create and destroy communicators to maintain a mapping between global ranks (rank number within `MPI_COMM_WORLD`) and local ranks (rank number within communicators created by users). This mapping is necessary because PERUSE events are reported with local ranks, while profiling results should be described with global ranks for the ease of analysis.

PFProf is provided in the form of a shared library, which can be integrated into applications either when building the application or running the application. The preferred way is to use run-time integration, because it requires neither recompilation nor relinking of the application. The `LD_PRELOAD` environment variable is used to load the shared library before the execution of the application.

A visualization of a traffic matrix obtained from running MILC with 128 processes is shown in Fig. 2. This visualization clearly reveals the spatial locality and sparsity of the communication pattern.

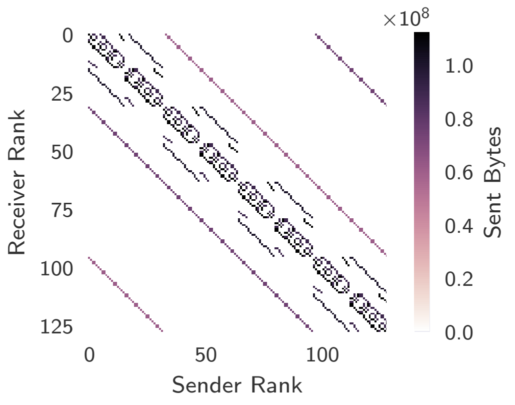


Figure 2: Traffic Matrix for MILC

C. *PFSim (Interconnect Simulator)*

PFSim uses a set of communication patterns of applications and a cluster configuration as its input and then simulates the packet flow generated by the applications. The packet flow is aggregated per link to compute the traffic load of each link. The simulated traffic load of links can be summarized into statistics for quantitative analysis or visualized. Using these outputs from PFSim, users can locate hot-spots and assess load imbalance in the interconnect. These insights on the interconnect can be useful for designing better algorithms for controlling the packet flow in application-aware dynamic interconnects.

PFSim is based on a discrete-event simulation model. Under this model, each event holds an attribute indicating when it will occur. Events are stored in a priority queue in an increasing manner by the time they occur. There are different type of events representing a change of state in the simulator such as: a job arrived, a job started, a job finished, *etc*. At the beginning of the main loop, the earliest occurring event is popped from the event queue. Then, based on the type of the event, the corresponding event handler is called. An event handler may schedule new events. This loop is repeated until the event queue is empty.

Figure 3 illustrates how PFSim works. The simulation scenario file defines various configurations for a simulation run. This file defines the cluster configuration to use and the set of jobs. Moreover, algorithms that control the execution and communication of jobs are specified as shown in Table I. An example of a simulation scenario file is shown in Listing 1. The cluster configuration file defines the topology of the interconnect, the capacity of links, and the number of processing elements for each computing node. This file is described in GraphML [18], an XML-based markup language for graphs. Popular graph visualization tools such as Cytoscape and Gephi can be used to view and edit GraphML files. Communication pattern files are obtained from applications using PFProf described in the previous section IV-B.

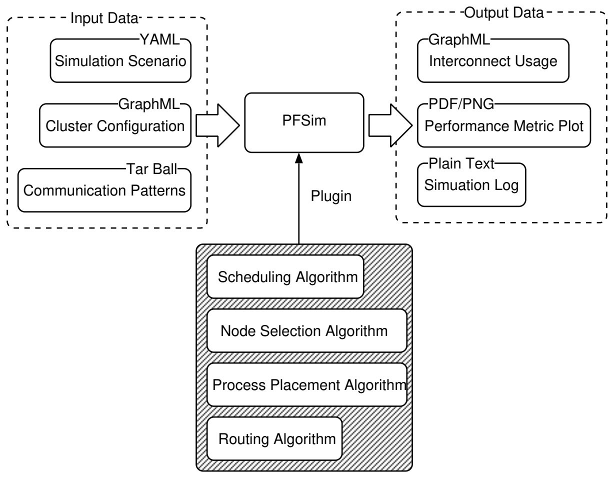


Figure 3: Block Diagram of PFSim

Each configuration value can be a list of values. In that case, the simulation is executed multiple times, each time with a different combination of configuration values until all combinations are completed.

PFSim can create a snapshot of the traffic distribution in the interconnect at an arbitrary time and output is as a

Table I: List of Configurable Algorithms

Algorithm	Description
Job Scheduling	Selects the job to execute from the job queue. (*e.g.* FCFS, Backfill)
Node Selection	Selects which computing nodes to assign for a job. (*e.g.* Linear, Random, Topology-aware algorithms)
Process Placement	Determines on which computing node to place a process. (*e.g.* Block, Cyclic, Application-aware algorithms)
Routing	Computes a route between a pair of processes. (*e.g.* D-mod-K, S-mod-K, Random, Dynamic algorithms)

GraphML file. By visualizing the acquired GraphML using the aforementioned graph visualization tools, users can intuitively locate bottleneck links and load imbalance. Furthermore, the traffic load of links can be summarized into statistics such as maximum, minimum, average, variance and plotted as graphs.

```
topology: topologies/milk.graphml
output: output/milk-cg-dmodk
algorithms:
  scheduler:
    - pfsim.scheduler.FCFSScheduler
  node_selector:
    - pfsim.node_selector.LinearNodeSelector
    - pfsim.node_selector.RandomNodeSelector
  process_mapper:
    - pfsim.process_mapper.LinearProcessMapper
    - pfsim.process_mapper.CyclicProcessMapper
  router:
    - pfsim.router.DmodKRouter
    - pfsim.router.GreedyRouter
    - pfsim.router.GreedyRouter2
jobs:
  - submit:
      distribution:
           pfsim.math.ExponentialDistribution
      params:
        lambd: 0.1
    trace: traces/cg-c-128.tar.gz
...
```

Listing 1: Example of a Simulation Scenario

Figure 4 illustrates the life cycle of a simulated job. On arrival of a new job, the job is initially enqueued to the job queue. As soon as there are enough unallocated computing nodes to execute a job, the scheduling algorithm determines the job to be executed next and pops it from the job queue. Subsequently, the node selection algorithm picks a set of computing nodes that can satisfy the requested number of processes by the job. Then, the process placement algorithm determines on which computing node to run each process of the job. After the process placement is completed, the routing algorithm computes and allocates routes for all communicating pairs of processes. To allow users to implement application-aware dynamic routings, additional information is supplied to the routing algorithm in addition to the source/destination process pair. The information includes process placement, communication patterns and interconnect usage. For each allocated route, the traffic on the link contained in the route is increased. After all routes are computed, PFSim waits until the job has finished. Then, the traffic for each link that was utilized by the job is decreased. Finally, computing nodes are deallocated and the job is marked as completed.

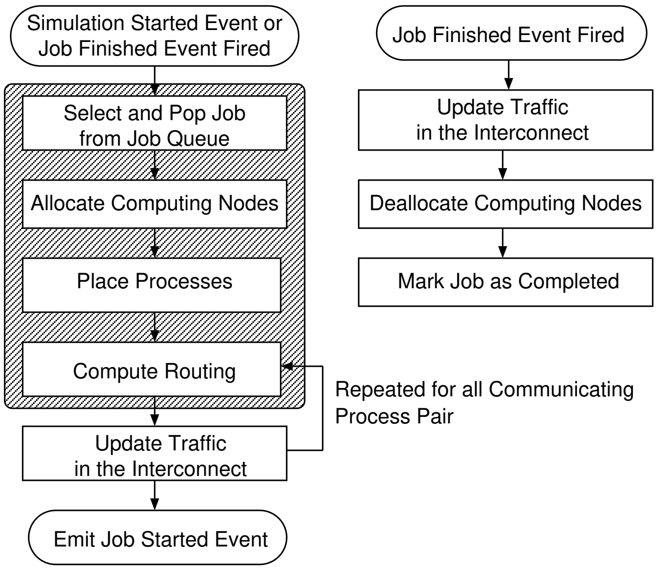

Simulation Started Event or Job Finished Event Fired
Select and Pop Job from Job Queue
Allocate Computing Nodes
Place Processes
Compute Routing
Repeated for all Communicating Process Pair
Update Traffic in the Interconnect
Emit Job Started Event
Job Finished Event Fired
Update Traffic in the Interconnect
Deallocate Computing Nodes
Mark Job as Completed

Figure 4: Life Cycle of a Simulated Job

V. Evaluation

In this section, we first simulate the traffic load of a fat-tree interconnect when using static interconnect control and dynamic interconnect control. Subsequently, benchmark results obtained from a physical cluster are used to investigate the impact of traffic load on the application performance. Lastly, we assess the overhead incurred by our profiler.

A. Simulation Results

In this experiment, communication-intensive MPI applications were executed on our simulator. The maximum traffic load observed on links composing the interconnect was compared in both cases of static interconnect control and dynamic interconnect control. The maximum traffic load observed on the links was used as an indicator of the communication performance of an application. In most cases, a hot spot link can slow down the whole application, because when collective communication or synchronization is performed by an application, every process needs to wait until the slow communication crossing the hot spot link completes. Therefore, mitigating the traffic load on the hot spot link could improve the performance of the application.

The simulated cluster was modeled after a physical cluster installed at our institution. It was composed of 20 computing nodes, each equipped with 8 cores. Computing nodes were interconnected with a fat-tree topology as illustrated in Fig. 5.

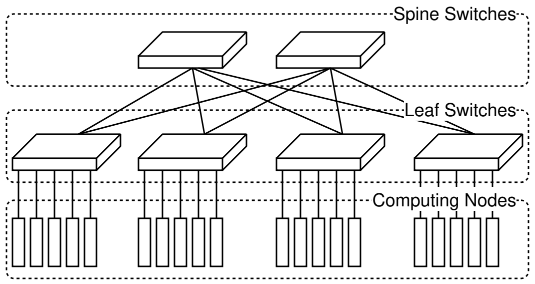


Figure 5: Simulated Cluster with Fat-Tree Interconnect

Two applications were selected as representatives of communication-intensive applications. The first one is the CG benchmark from the NAS Parallel Benchmark Suite [19]. The CG benchmark estimates the largest eigenvalue of a sparse matrix using the inverse power method. Internally it uses the conjugate gradient method, which frequently appears in irregular mesh applications. The second application (`ks_imp_dyn`) is from MIMD Lattice Computation (MILC) [20], a collection of applications used to study Quantum Chromodynamics (QCD). We used the input dataset provided by NERSC as a part of the NERSC MILC benchmark. These two applications were executed with 128 MPI processes. Thread parallelism was not put in use (*i.e.* a flat MPI model was adopted).

To analyze the effect of dynamic interconnect control, simulations were carried out either using static routing or dynamic routing. Furthermore, in order to investigate the impact of node selection and process placement to the traffic load, the node selection algorithm and process placement algorithm were also changed. As a result, exhaustive combinations of two node selection algorithms, two process placement algorithms and two routing algorithms were investigated with the scheduling algorithm fixed. Below are the descriptions of the algorithms used in this experiment:

- Scheduling: A simple *First-Come First-Served (FCFS)* scheduling without backfilling was adopted.
- Node Selection: Either *linear* or *random* node selection was adopted. Linear node selection assumes that computing nodes are lined up in a one-dimensional array and minimizes the fragmentation of allocation. This is essentially the same as the default node selection policy of Slurm. Random node selection randomly selects computing nodes. This algorithm simulates a situation where the allocation of computing nodes is highly fragmented.
- Process Placement: Either *block* or *cyclic* process placement was adopted. Block process placement assigns rank i to the $\lfloor i/c \rfloor$-th computing node where c represents the number of cores per node. Cyclic process placement assigns rank i to the (i mod n)-th computing node where n denotes the number of computing nodes.
- Routing: Either *D-mod-K* routing or a *dynamic* routing was adopted. Destination-modulo-K (D-mod-K) routing is a popular static load balancing routing algorithm that distributes packet flow over multiple paths based on the destination address of the packet. The dynamic routing algorithm implemented here computes and allocates routes from the heaviest communicating process pair. A route is computed to minimize the traffic of the maximum-traffic link in the path.

Under this condition, we measured and compared the maximum traffic load observed on links through the simulation. Figure 6 shows the simulation results in the NAS CG benchmark. In this graph, red bars represent the results of D-mod-K routing while blue bars represent the results of dynamic routing. The vertical axis represents the simulated maximum traffic load normalized by the maximum traffic load when linear node selection, block process placement and D-mod-K routing is adopted.

What stands out in Fig. 6 is that dynamic routing consistently achieves lower traffic load compared to static D-mod-K routing. The reduction of traffic load was largest when linear node selection and block process placement were adopted. Under this combination of node selection and the process placement algorithm, dynamic routing slashed the maximum traffic load by 50% compared to D-mod-K routing. In addition, the graph reveals that cyclic process placement always increased maximum traffic load compared to block process placement because neighboring ranks were placed on different computing nodes despite the locality of the communication pattern.

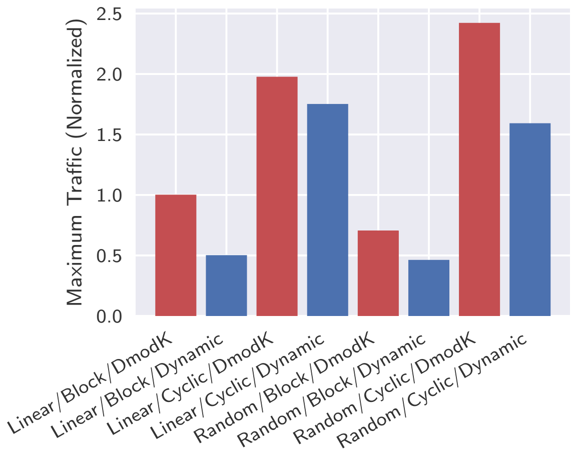


Figure 6: Comparison of Maximum Traffic (NAS CG)

Figure 7 shows the result in the case of MILC. The graph reveals that dynamic routing outperforms D-mod-K routing. In this case, the reduction of the link load was largest when random node selection and cyclic process placement was adopted. When using linear node selection and block process placement, reduction of the maximum link load was 18%.

B. Benchmark Results

To investigate the impact of traffic load on the application performance of an actual environment, we reproduced the configuration described in the previous section V-A on a physical cluster and then measured the execution time of each benchmark. This cluster was equipped with switches that support OpenFlow, which is a de facto standard implementation

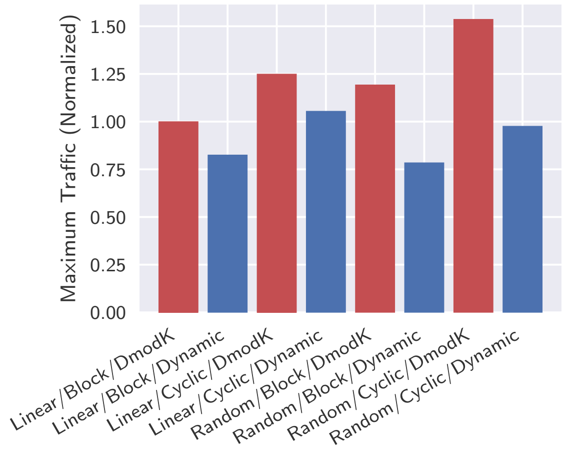
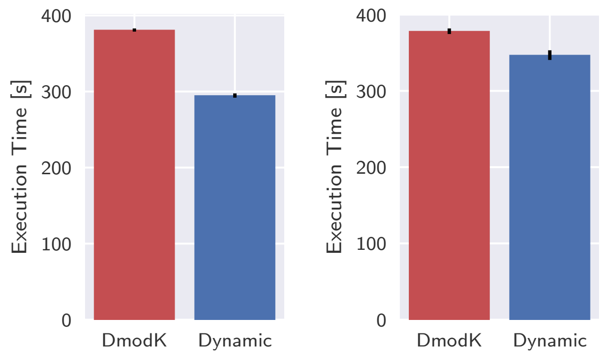


Figure 7: Comparison of Maximum Traffic (MILC)

of SDN. The routing algorithms were implemented based on OpenFlow. In this experiment, linear node selection and block process placement was adopted. The average execution time of 10 runs was compared when using D-mod-K routing and dynamic routing. Figure 8 (a) shows the comparison for the NAS CG benchmark. The graph indicates that the use of dynamic routing reduced the execution time of the benchmark to 23%. Figure 8 (b) shows the result for MILC. In this case, approximately 8% was reduced in execution time.

These results suggest that application performance is actually improved by alleviating the traffic load on the hot spot link. This suggestion implies that researchers of dynamic interconnects can take advantage of our toolset to simulate different packet flow controlling algorithms and assess their performance improvement effect on real-world applications by using indicators such as maximum traffic load.



(a) NAS CG (b) MILC

Figure 8: Comparison of Execution Time

C. Profiling Overhead

In this experiment, the performance of point-to-point communication between two processes with and without PFProf were compared to inspect the overhead incurred by the profiler. OSU Micro Benchmark [21] was used to measure the throughput and latency of point-to-point communication between two processes for varying message sizes. The comparison of throughput is shown in Fig. 9. For messages larger than 1KB, the overhead was ignorable. For messages smaller than 1KB, up to 30% of overhead was incurred. Benchmark results for latency, as shown in Fig. 10, suggest that there is almost no overhead for latency.

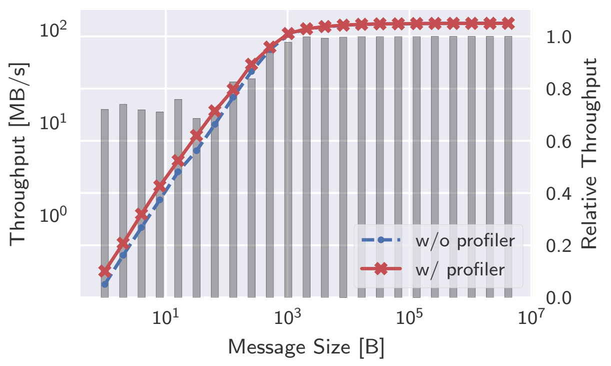


Figure 9: Throughput of MPI_Send/Recv Between Two Nodes

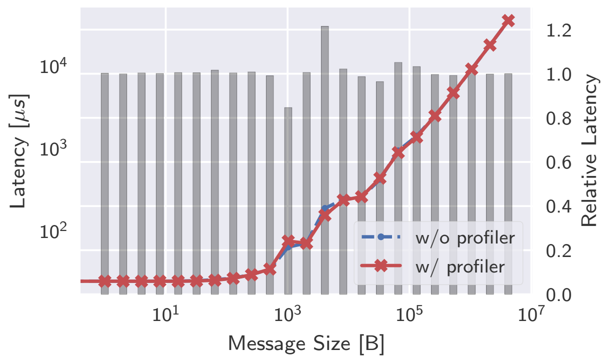


Figure 10: Latency of MPI_Send/Recv Between Two Nodes

VI. Conclusion

This paper described the design and implementation of PFAnalyzer, a toolset for analyzing the performance characteristics of application-aware dynamic interconnects. PFAnalyzer is composed of PFProf, a profiler to extract communication patterns from applications, and PFSim, a simulator capable of simulating application-aware dynamic interconnects. PFSim takes a set of communication patterns of applications and a cluster configuration as its input and then simulates the traffic on each link of the interconnect. Our evaluation shows how dynamically controlling the interconnects can reduce congestion and potentially improve the performance of applications.

Further work is necessary to investigate the performance characteristics of dynamic interconnects on large-scale and highly parallel clusters. Moreover, we plan to implement application-aware node selection and process placement algorithms on PFSim. The impact of such application-aware algorithms on the performance of dynamic interconnects should be evaluated.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP26330145 and JP17KT0083. This research was also supported in part by the "Program for Leading Graduate Schools" of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, 2015. [Online]. Available: http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[2] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface, Third Edition*. The MIT Press, 2014, isbn: 9780262527392.

[3] InfiniBand Trade Association, *InfiniBand Architecture Specification Release 1.3*, 2015.

[4] S. Date, H. Abe, D. Khureltulga, K. Takahashi, Y. Kido, Y. Watashiba, P. U-chupala, K. Ichikawa, H. Yamanaka, E. Kawai, and S. Shimojo, "SDN-accelerated HPC Infrastructure for Scientific Research", *International Journal of Information Technology*, vol. 22, no. 1, 2016.

[5] K. Takahashi, D. Khureltulga, Y. Watashiba, Y. Kido, S. Date, and S. Shimojo, "Performance Evaluation of SDN-enhanced MPI_Allreduce on a Cluster System with Fat-tree Interconnect", in *2014 International Conference on High Performance Computing & Simulation (HPCS 2014)*, 2014, pp. 784–792. doi: 10.1109/HPCSim.2014.6903768.

[6] K. Dashdavaa, S. Date, H. Yamanaka, E. Kawai, Y. Watashiba, K. Ichikawa, H. Abe, and S. Shimojo, "Architecture of a High-Speed MPI_Bcast Leveraging Software-Defined Network", in *6th Workshop on UnConventional High Performance Computing (UCHPC2013)*, 2014, pp. 885–894. doi: 10.1007/978-3-642-54420-0_86.

[7] Open Networking Foundation, *Software-Defined Networking: The New Norm for Networks*, 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[8] T. Schneider and T. Hoefler, "ORCS: An Oblivious Routing Congestion Simulator", *Indiana University, Computer*, no. 675, 2009.

[9] M. M. Tikir, M. A. Laurenzano, L. Carrington, and A. Snavely, "PSINS: An Open Source Event Tracer and Execution Simulator", in *Department of Defense High Performance Computing Modernization Program - Users Group Conference (HPCMP-UGC)*, 2009, pp. 444–449. doi: 10.1109/HPCMP-UGC.2009.73.

[10] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim: Simulating Large-scale Applications in the LogGOPS Model", in *19th International Symposium on High Performance Distributed Computing (HPDC '10)*, 2010, pp. 597–604. doi: 10.1145/1851476.1851564.

[11] E. Jo, D. Pan, J. Liu, and L. Butler, "A Simulation and Emulation Study of SDN-based Multipath Routing for Fat-tree Data Center Networks", in *2014 Winter Simulation Conference (WSC 2014)*, 2015, pp. 3072–3083. doi: 10.1109/WSC.2014.7020145.

[12] H. Subramoni, A. M. Augustine, M. Arnold, J. Perkins, X. Lu, K. Hamidouche, and D. K. Panda, "INAM2: InfiniBand Network Analysis and Monitoring with MPI", in *International Conference on High Performance Computing (ISC)*, 2016, pp. 300–320. doi: 10.1007/978-3-319-41321-1_16.

[13] L. Alawneh, A. Hamou-Lhadj, and J. Hassine, "Segmenting Large Traces of Inter-process Communication with a Focus on High Performance Computing Systems", *Journal of Systems and Software*, vol. 120, pp. 1–16, 2016. doi: 10.1016/j.jss.2016.06.067.

[14] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir", in *Tools for High Performance Computing*, 2012, pp. 79–91. doi: 10.1007/978-3-642-31476-6_7.

[15] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The Vampir Performance Analysis Tool-Set", in *Tools for High Performance Computing*, 2008, pp. 139–155. doi: 10.1007/978-3-540-68564-7_9.

[16] S. S. Shende and A. D. Malony, "The Tau Parallel Performance System", *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006. doi: 10.1177/1094342006064482.

[17] T. Jones, *MPI PERUSE: An MPI Extension for Revealing Unexposed Implementation Information*, 2006.

[18] U. Brandes, M. Eiglsperger, J. Lerner, and C. Pich, "Graph Markup Language (GraphML)", *Handbook of Graph Drawing and Visualization*, pp. 517–541, 2013.

[19] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS Parallel Benchmarks", *International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991. doi: 10.1177/109434209100500306.

[20] MIMD Lattice Computation Collaboration, *MIMD Lattice Computation (MILC) Collaboration Home Page*. [Online]. Available: http://physics.indiana.edu/~sg/milc.html.

[21] Ohio State University, *OSU Micro Benchmark*. [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/.