Scalable Fine-Grained Gang Scheduling for HPC Systems with Unreliable Broadcast Synchronization Mechanisms

Hiroki Ohtsuji*
ohtsuji.hiroki@fujitsu.com
Fujitsu Ltd.
Kawasaki, Kanagawa, Japan

Erika Hayashi
Fujitsu Ltd.
Kawasaki, Kanagawa, Japan

Reika Kinoshita
Fujitsu Ltd.
Kawasaki, Kanagawa, Japan

Masahiro Miwa
Fujitsu Ltd.
Kawasaki, Kanagawa, Japan

Eiji Yoshida
Fujitsu Ltd.
Kawasaki, Kanagawa, Japan

ABSTRACT

The demand for interactivity on HPC systems is increasing, primarily driven by new HPC users from the AI/ML research area. Traditional HPC users are accustomed to waiting for job execution on a batch scheduling system while new users prefer an interactive terminal such as Jupyter Notebook. To address these evolving requirements, enhancing interactivity is essential. Fine-grained gang scheduling is one potential solution for this problem. This poster presents a scalable inter-node synchronization mechanism that facilitates well-time-aligned synchronization message delivery through broadcast communication for fine-grained gang scheduling in HPC systems. The mechanism improved the application performance by 2.7 times compared to the existing implementation, when concurrently running two parallel applications on 128 computing nodes using a 500ms time slice duration.

1 INTRODUCTION

High-Performance Computing (HPC) systems are primarily operated using batch queue systems, which are managed using job schedulers. These systems typically employ a First-In-First-Out (FIFO) approach, resulting in some waiting time even for short duration jobs. Existing job scheduling implementations prioritize short duration and low node count jobs through a backfilling mechanism. Backfilling is theoretically effective in a congested environment; however, it requires accurate job execution time estimation by users for jobs in the queue [3]. Users tend to specify longer estimated times for jobs to avoid unintentional job termination. Too long job estimation time obstruct backfilling because job schedulers determine that job does not fit in the available space at an earlier time. The typical HPC system can limited interactivity because of the limitation of backfilling.

The increasing scale of AI/ML applications is changing the demand from users and creating pressure to provide more interactivity in HPC systems. Users in the traditional HPC community are accustomed to waiting for job scheduling while new users from AI/ML field are using cloud computing environment, that can provide resources instantly. They often use interactive terminal like Jupyter Notebook, where waiting hours for execution, even for short tasks, is uncommon.

There are two approaches to improving interactivity in HPC systems: space sharing and time sharing. Space sharing divides computing resources into several parts and provides them to multiple jobs or users. Most paralel applications with MPI do not support rescaling of running application, so space sharing does not meet the requirement to run scalable applications. Time sharing allows multiple jobs or users to share the same computing resources in a time-shared manner. The system switches between jobs on the resource at fixed intervals (time slices) for concurrent execution.

Ordinary time sharing execution is not suitable for parallel applications because those applications require synchronous messaging and reduction operations, that require that all resources are available at the same time. Gang scheduling[4] is a solution for running multiple parallel applications concurrently on limited computing resources. It synchronizes the states of the execution across the computing resources allocated to each job. Historically, the adoption of gang scheduling in real-world systems has been limited due to the lack of demand, risk of memory shortages, and the reduced execution efficiency with large scale applications. The first issue is solved by changes in demand. The second issue becomes relatively non-critical because of the growing capacity of memory devices [5]. The third issue is caused by synchronization jitters on multiple computing nodes. This paper proposes a new approach to address the third issue by improving the efficiency of gang scheduling for large-scale applications in HPC systems by using the scalable high precision inter-node synchronization mechanism.

2 RELATED WORK

The impact of synchronization methods on performance in gang scheduling has been evaluated and analyzed in [1]. Although the era and system configurations in which this paper was written are different, the requirement for task switching to be performed with high temporal accuracy remains unchanged. This poster describes the implementation and performance evaluation in a large-scale cluster environment. While not directly related to schedulers, [2] demonstrates that data drops do not realistically occur in large-scale clusters connected with InfiniBand, even when using Unreliable messages. This is an important background information for our study because this research aims to perform scalable job switching for gang scheduling by using unreliable broadcast messages. Widely used scheduler like Slurm also has a gang scheduling implementation; however, it does not support the time slice shorter than 5 seconds. A lower time slice is better for interactivity and our target is less than 0.5 seconds.

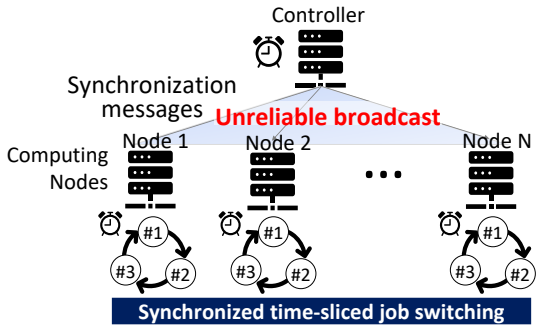


Figure 1: Each computing node switches jobs based on the synchronization signal received from the control server.

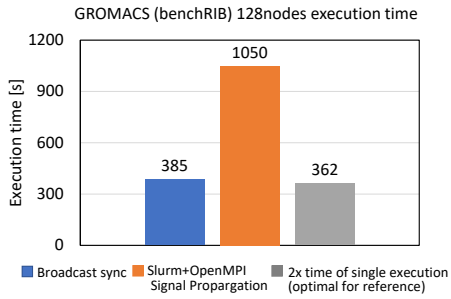


Figure 2: Performance comparison between the proposed synchronization mechanism and OpenMPI signal propagation.

3 GANG SCHEDULING WITH HIGH-PRECISION INTER-NODE SYNCHRONIZATION

3.1 Design

In this poster, we propose a scalable inter-node job switching mechanism for gang scheduling in HPC systems. The architecture of the proposed system is illustrated in Figure 1. We created a partition with oversubscription enabled within the Slurm environment, allowing multiple jobs to operate in a time-sharing manner on the same resources. It is important to note that we did not utilize the gang scheduling functionality provided by Slurm. Instead, we implemented an external node synchronization job switching mechanism for this system.

The implementation can be divided into several parts: a controller that sends synchronization messages using broadcast communication, a mapping of the job list in Slurm and the process IDs on each computing node, and an agent that switches jobs when synchronization messages are received on each computing node. Job switching is performed by sending a SIGSTOP signal to the processes of the jobs to be suspended and a SIGCONT signal to the jobs to be resumed, based on the correspondence between job IDs and PIDs. The job switching signal is transmitted as a UDP broadcast packet over IP over InfiniBand network. Currently, synchronization messages only contain sequence numbers. The agent program switches running jobs based on the received sequence numbers because this is the prototype to evaluate the effectiveness of the broadcast synchronization mechanism. There are several algorithms to determine the combination of jobs to run concurrently, and implementing them in the system is our future work.

The novelty of this research lies in the use of unreliable communication, such as UDP, for controlling job and process switching, which is typically performed using highly reliable and unicast communication, such as TCP. Unlike unicast communication that requires sequential operations for broadcasting, it is expected that the synchronization will be highly accurate. Regarding the concern of data loss of broadcast communication, according to our experiments, no packet loss was observed on the 1000-node InfiniBand connected cluster systems. Even if rare data loss occurs, it would only result in a single job switch not being performed, which is not a critical issue. The expected performance degradation caused by data loss is almost the same as the packet loss ratio, which is typically very low on HPC networks.

3.2 Evaluation

In this section, we present the evaluation of the gang scheduling with the proposed synchronization mechanism. The synchronization mechanism is integrated into the Slurm 22.05 environment. The time-slice duration is 500ms for all evaluations. Please refer to the poster for the detailed information of environment.

The evaluation was carried out by time-slicing the execution of two simultaneous jobs and recording their execution times. The target applications were Intel MPI benchmarks and GROMACS (benchRIB). Please refer to the poster for MPI communication performance. We compared the performance of the proposed gang scheduling with the synchronization mechanism to that of the signal propagation mechanism provided by OpenMPI. We utilized the signal propagation mechanism of slurm by sending signals only to processes on the first node of each partition.

Figure 2 shows a graph of the evaluation results, where the vertical axis represents the execution time and the horizontal axis represents the time slice duration. The graph illustrates the performance of both the proposed mechanism (Broadcast sync.) and the signal propagation mechanism of Slurm/OpenMPI. The proposed mechanism demonstrated shorter execution times compared to the Slurm's signal propagation implementation and there was only 6.3% overhead compared to the doubled time of single job execution.

4 CONCLUSION

In this research paper, we propose a new synchronization mechanism for scalable gang scheduling that utilizes unreliable broadcast messages. Through evaluations, our proposed mechanism exhibits superior performance compared to existing implementations. Furthermore, our approach maintains high performance and low overhead (6.3%) even as the number of nodes increases up to 128, indicating its high scalability. Supporting GPUs is future work; however, this method can be applied to GPU clusters because node-level synchronization is a common issue. This poster presents the details of our proposed gang scheduling mechanism and the results of our evaluations, highlighting its potential for enhancing the interactivity of parallel computing systems.

REFERENCES

[1] Anoop Gupta, Andrew Tucker, and Shigeru Urushibara. 1991. The Impact of Operating System Scheduling Policies and Synchronization Methods of Performance of Parallel Applications. *SIGMETRICS Perform. Eval. Rev.* 19, 1 (apr 1991), 120–132. https://doi.org/10.1145/107972.107985

[2] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 185–201. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kalia

[3] Ahuva W. Mu'alem and Dror G. Feitelson. 2001. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE transactions on parallel and distributed systems* 12, 6 (2001), 529–543.

[4] John K Ousterhout et al. 1982. Scheduling Techniques for Concurrent Systems.. In *ICDCS*, Vol. 82. 22–30.

[5] Ivy Peng, Ian Karlin, Maya Gokhale, Kathleen Shoga, Matthew Legendre, and Todd Gamblin. 2022. A Holistic View of Memory Utilization on HPC Systems: Current and Future Trends. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '21)*. Association for Computing Machinery, New York, NY, USA, Article 14, 11 pages. https://doi.org/10.1145/3488423.3519336