Paving the Road to Exascale with Many-Task Computing

Ke Wang
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA
kwang22@hawk.iit.edu

Anupam Rajendran
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA
arajend5@hawk.iit.edu

Kevin Brandstatter
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA
kbrandst@hawk.iit.edu

Zhao Zhang
Department of Computer Science
University of Chicago
Chicago, IL, USA
zhaozhang@uchicago.edu

Ioan Raicu
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA
iraicu@cs.iit.edu

ABSTRACT
Exascale systems will bring significant challenges. This work attempts to addresses them through the Many-Task Computing (MTC) paradigm, by delivering data-aware job scheduling systems and fully asynchronous distributed architectures. MTC applications are structured as DAG graphs of tasks, with dependencies forming the edges. The asynchronous nature of MTC makes it more resilient than traditional HPC approaches as the system MTTF decreases. Future highly parallelized hardware is well suited for achieving high throughput with large-scale MTC applications. This work proposes a distributed MTC execution fabric for exascale, MATRIX, which adopts work stealing to achieve load balance. Work stealing was studied through SimMatrix, a scalable simulator, up to exascale with millions of nodes, billions of cores, and trillions of tasks.

1. INTRODUCTION
Predictions are that 2019 will be the year of exascale with millions of nodes and billions of threads of execution [1]. With billions of threads of concurrency, we expect that the applications running on an Exascale machine would be decomposed with large number of tasks with very finer granularity in both size and duration, along with large volume of data.

Driven by the embarrassingly parallel tasks and the quantity of data, Many-Task Computing (MTC) was proposed [2] to bridge the gap between HPC and HTC. Many MTC applications are structured as graphs of discrete tasks, with input and output dependencies forming the graph edges. MTC applications often demand a short time to solution, may be communication intensive or data intensive [3]. For many applications, a graph of distinct tasks is a natural way to conceptualize the computation. Examples of MTC systems are various workflow systems, such as Swift [4], MapReduce systems, such as MapReduce [5], distributed run-time systems such as Charm++ [6], and light-weight task execution frameworks , such as Falkon [7], Sparrow [8]). We believe that the task execution framework of MTC would be distributed. However, with distributed architecture, issues can arise in balancing loads across all servers.

Load balancing refers to distribute workloads evenly across nodes of a supercomputer. This work adopts work stealing [9] to achieve distributed load balancing, where the idle processors steal tasks from the heavily-loaded ones. We explore the performance of work stealing in SimMatrix and MATRIX, which are simulator and real system of task execution framework, respectively.

2. RELATED WORK
The earliest batch job schedulers are Condor [10], Slurm [11]. All these systems target as the HPC or HTC applications, and lack the granularity of scheduling jobs at node/core level, making them hard to be applied to the MTC applications. What's more, the centralized dispatcher in these systems suffers scalability and reliability issues. Falkon [7] is a light-weight task execution framework with both centralized and hierarchical architectures for MTC workload, and although it scaled and performed several orders magnitude better than the traditional batch schedulers, it even cannot scale to petascale systems [12]. Sparrow [8] is another hierarchical task execution framework targeting at sub-second tasks. However the Java-based framework is very hard to be deployed on supercomputers.

3. TASK EXECUTION FRAMEWORK
3.1 SimMatrix
SimMatrix [13] is a simulator for MTC execution fabric at exascale. The architectures of SimMatrix are shown in Figure 1.

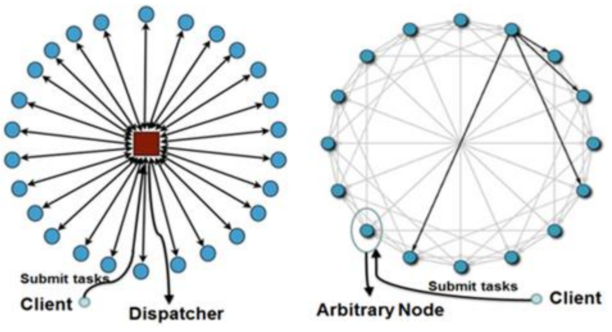

Figure 1: SimMATRIX architectures

For simplicity, we assign consecutive integer numbers as the node ids, ranging from 0 to the number of node N-1. SimMatrix supports the granularity of scheduling at the node/core level at extreme scales. The system could be centralized (Figure 1 left part), where a single dispatcher maintains a task queue and manages the task submission, task assignment, and task execution state updates. It could also be distributed (Figure 1 right part), where each computing node maintains a task execution framework, and they cooperate with each other to achieve load balancing through work stealing technique.

3.2 MATRIX

MATRIX is a distributed MTC execution framework that implements work stealing technique. MATRIX uses ZHT [14], a distributed zero hop key-value store, to manage job metadata, to submit tasks, and to monitor the task execution progress. The components of MATRIX and the communication signals among them are shown in Figure 2. The client is a benchmarking tool that issues request to generate a set of tasks to be executed. The client has a task dispatcher that helps submit workload to the compute nodes. A compute node can also be referred as worker node that has a task execution unit along with a ZHT server for managing the metadata of every task.

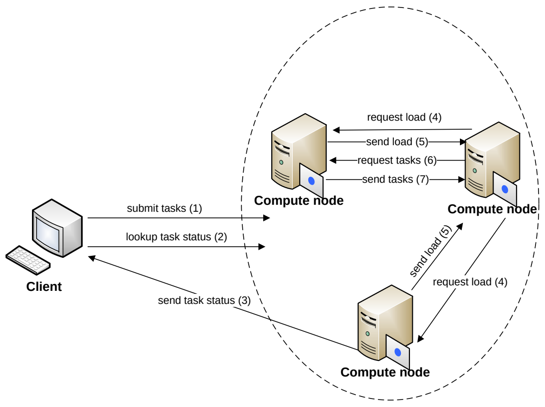

Figure 2: MATRIX components and communication signals

Upon request from the client, with the help of ZHT, the task dispatcher initializes the workload of given type and submits tasks to one arbitrary node, or to all the nodes in a balanced distribution. All compute nodes execute tasks, and distribute the workload among them adaptively to achieve load balancing via the work stealing algorithm. The client periodically monitors the status of workload until all the tasks are executed.

4. PERFORMANCE EVALUATION

SimMatrix runs on a single-node machine, and is validated against MATRIX. Through SimMatrix, we explore important parameters of work stealing that are important to the performance, such as the number of tasks to steal, the number of neighbors to steal tasks, static/dynamic neighbors and poll interval. We have scaled SimMatrix to exascale with millions of nodes, billions of cores, and hundreds of billions tasks. In Figure 3, we show that work stealing is the approach to exascale to achieve distributed load balancing. Each node is configured to have 1000 cores, and the number of tasks is ten times of the number of cores. We see that even at exascale with work stealing, SimMatrix can achieve about 90% efficiency.

5. CONCLUSION & FUTURE WORK

Distributed load balancing is critical for designing job schedulers. Work stealing is a potential technique to achieve distributed load balancing across many concurrent threads of execution. We will continue to develop the MATRIX system, and plan to test it on the newly built IBM Blue Gene/Q supercomputer at a full 768K-core (3M hardware threads) scale. MATRIX will also be integrated with other projects, such as MapReduce and FusionFS file system to support data-aware scheduling, and large scale programming runtime systems, such as Charm++ to explore different load balancing techniques.

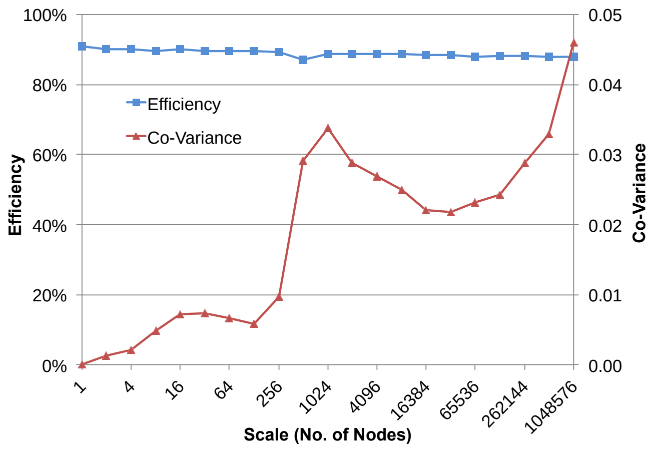

Figure 3: Running SimMatrix at exascale

6. REFERENCES

[1] V. Sarkar, et al. “ExaScale Software Study: Software Challenges in Extreme Scale Systems”, ExaScale Computing Study, DARPA IPTO, 2009.

[2] I. Raicu, Y. Zhao, I. Foster, “Many-Task Computing for Grids and Supercomputers,” 1st IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS) 2008.

[3] I. Raicu et. al. “Towards Data Intensive Many-Task Computing”, book chapter in Data Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management, IGI Global Publishers, 2011.

[4] Y. Zhao et. al. “Swift: Fast, Reliable, Loosely Coupled Parallel Computation,” IEEE Workshop on Scientific Workflows 2007.

[5] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Comm. ACM, Jan. 2008, pp. 107-113.

[6] G. Zhang, et. al, “Hierarchical Load Balancing for Charm++ Applications on Large Supercomputers,” In Proceedings of the 2010 39th International Conference on Parallel Processing Workshops, ICPPW 10, pages 436-444, Washington, DC, USA, 2010.

[7] I. Raicu, et. al. “Falkon: A Fast and Light-weight tasK executiON Framework,” IEEE/ACM SC 2007.

[8] K. Ousterhout et. al. “Batch Sampling: Low Overhead Scheduling for Sub-Second Prallel Jobs.” University of California, Berkeley, 2012.

[9] R. D. Blumofe et. al. “Scheduling multithreaded computations by work stealing,” In Proc. 35th FOCS, pages 356–368, Nov. 1994.

[10] J. Frey et. al. “Condor-G: A Computation Management Agent for Multi-Institutional Grids,” Cluster Computing, 2002

[11] M. A. Jette et. al, Slurm: Simple linux utility for resource management. Proceedings of Job Scheduling Strategies for Prarallel Procesing (JSSPP) 2003 (2002), Springer-Verlag, pp. 44-60.

[12] I. Raicu, et. al. “Toward Loosely Coupled Programming on Petascale Systems,” IEEE SC 2008.

[13] K. Wang, K. Brandstatter, I. Raicu. “SimMatrix: Simulator for MAny-Task computing execution fabRIc at eXascales”, ACM HPC 2013.

[14] T. Li, et. al. “ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table”, 27th IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2013.