MATRIX: MAny-Task computing execution fabRIc at eXascale

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

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

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

ABSTRACT

Efficiently scheduling large number of jobs over large-scale distributed systems is critical in achieving high system utilization and throughput. Most of current job management systems (JMS) have centralized Master/Slaves architecture that has inherent limitations, such as scalability issues at extreme scales (e.g. petascales and beyond), and single point of failure. In designing the next generation distributed JMS, we must address new challenges such as load balancing. This paper presents MATRIX, a many-task computing execution fabric at exascale. MATRIX utilizes adaptive work stealing algorithm for distributed load balancing, and distributed hash tables for managing task metadata. MATRIX supports many-task computing (MTC) workloads with or without task dependencies in the execution of complex large-scale workflows. MATRIX has shown throughput as high as 54.4K tasks/sec at 4K-core scales running on an IBM Blue Gene/P supercomputer with sub-second sleep tasks (64ms).

Keywords

job scheduling, many-task computing, load balance

1. INTRODUCTION

With the dramatically increase of the scales of distributed systems and the finer granularity of jobs in both size and duration, it is urgent to develop distributed job schedulers that can deliver jobs several magnitudes faster than current centralized ones (e.g. Slurm [1], Condor [2]), where a server is managing the resource provisioning and job execution. 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, so that no one is overloaded. We believe that distributed load balancing techniques are potential approaches to extreme scale. This work adopts work stealing [3] to achieve distributed load balancing, where the idle processors steal tasks from the heavily-loaded ones. We explore the performance of work stealing in the MATRIX task execution framework.

This work is motivated by the Many-Task Computing (MTC) [4] paradigm which tries to bridges the gap between High Performance Computing (HPC) and High Throughput Computing (HTC). Many MTC applications are structured as graphs of discrete tasks, with explicit input and output dependencies forming the graph edges. MTC applications often demand a short time to solution, may be communication intensive or data intensive [5]. 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 [6], MapReduce systems, such as MapReduce [7], distributed run-time systems such as Charm++ [8], and light-weight task execution frameworks , such as Falkon [9], Sparrow [10]).

2. RELATED WORK

The earliest batch job schedulers are Condor [2], Slurm [1]. 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 [9] 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 [11]. Sparrow [10] 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. MATRIX

MATRIX is a distributed MTC execution framework that implements work stealing technique. MATRIX uses ZHT [12], a distributed zero hop key-value store, to manage job metadata, to submit tasks, and to monitor the task execution progress. We have a functional prototype implemented in C/C++, and have scaled it on a BG/P machine up to 4K-cores with good results.

3.1 MATRIX Architecture

The components of MATRIX and the communication signals among them are shown in Figure 1. 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. MATRIX supports single-core, single-node and multi-node tasks. It also supports task dependency, which means the orders of execution among a workload's tasks can be specified as a part of task description and MATRIX would guarantee the execution order. The workload would be represented as a Directed Acyclic Graph (DAG) where each vertex is a task and the edges specify the dependency. 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 (parameter space has been explored through SimMatrix simulator [13]). The client periodically monitors the status of workload until all the tasks are executed.

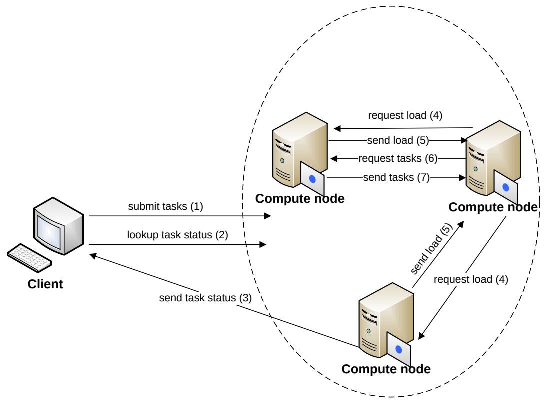

Figure 1: MATRIX components and communication signals

The worker node maintains three queues: wait queue, ready queue and complete queue, in the execution unit. The wait queue holds the incoming tasks that have dependency conditions to be satisfied. Once satisfied, the tasks are moved to the ready queue being executed in FIFO way. After a task is executed, it would be moved to the complete queue. Also, the execution unit is sending dependency satisfaction messages to ZHT server to notify all children tasks.

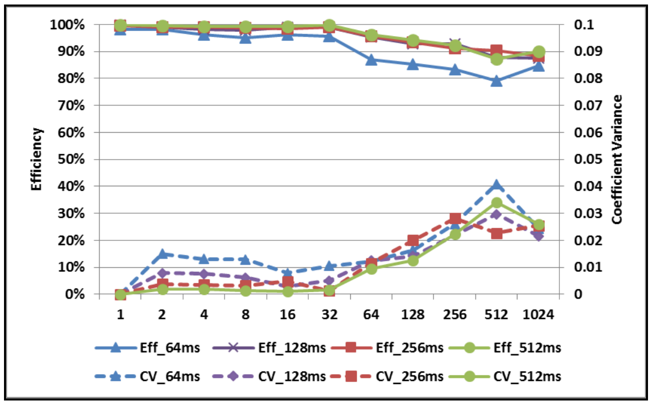

Figure 2: Running MATRIX for fine granular tasks

4. Performance Evaluation

We evaluate the performance of MATRIX with 4 different workloads (sleep tasks) on a BG/P machine up to 1K nodes (4K cores). The workloads are: Bag of Tasks (no dependency), Fan-Out DAG (tree-shaped dependency), Fan-In DAG (reverse tree-based dependency), and Pipeline DAG (multiple pipelines). The metrics are system efficiency (system utilization) and coefficient variance of number of tasks executed by each compute node. We show the evaluation of MATRIX with Bag of Tasks workloads.

In Figure 2, each node has 4 cores, and the number of tasks is 1000 times of the number of cores.We see that for very fine granular sub-second tasks, MATRIX acutally performs very well with 80%+ efficiency even at 1K-node scale. At 1K-node scale, for 64ms tasks, MATRIX achieves 85% efficency, meaning throughput of 54.4K tasks/sec (1024 * 1000 * 4 / 64 * 0.85). This throughput is several orders of magnitude larger than today’s batch schedulers.

5. Conclusion and Future Work

Large scale distributed systems require efficient job scheduling system to achieve high throughput and system utilization for small and shorter jobs. 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. The work stealing algorithm was implemented in a MATRIX, and a preliminary evaluation up to 4K-core scales was performed for different types of workloads with great results.

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++ [8] to explore different load balancing techniques.

6. REFERENCES

[1] 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.

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

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

[4] 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.

[5] 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.

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

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

[8] 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.

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

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

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

[12] 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.

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