MATRIX

Summary - This article gives the detailed explanation of many-task computing (MTC) on exascale systems with hundreds of thousands of nodes with up to a billion threads of execution. The chapter also address the problem of scheduling and throughput in exascale MTC applications through distributed scheduling architecture and load balancing technique (i.e. work stealing). MTC model require a fully distributed scheduling architecture. The chapter discusses about the lightweight simulator, SimMatrix, which simulates both centralized and distributed MTC task execution frameworks up to 1-million nodes, 1-billion cores, and 100 billion tasks, on both shared-memory many-core machine and distributed exascale billion-core machine. With the results obtained and insights gained a new distributed task execution framework was developed called MATRIX.

MATRIX is a real distributed task execution framework which implements the stealing work technique to achieve distributed load balancing and employs a distributed key-value store (i.e. ZHT) for task management data. MATRIX is deployed on an IBM Blue Gene/P (BG/P) supercomputer and 72 Amazon cloud running both independent tasks and tasks with dependencies specified as Direct Acyclic Graphs (DAG). The Adaptive Work Stealing Technique to achieve distributed load balancing technique on exascale systems has proven an efficient technique at the core level I a shared memory environment. In this pull based technique the ideal processor in the system randomly steals tasks form the overloaded one. In a fully distributed system every scheduler maintains a global membership list and is aware of every other scheduler and their tasks. Thus, the selection of candidate neighbor(victim) from which the ideal scheduler(thief) steals tasks could be static or dynamic. The components of the MATRIX are a scheduler, an executor and a ZHT server. A client is a benchmarking tool that issues request to generate a set of tasks and submits to any scheduler. The executor keeps on executing the tasks until the scheduler is left with no tasks. Whenever the scheduler has no more waiting tasks it starts stealing tasks from its neighboring scheduler(victim). MATRIX is developed in C++ and implemented to the client that generates tasks according to the input requirement, inserts the tasks dependencies to ZHT servers, submits tasks and monitors the execution progress.

A client can submit tasks to any arbitrary scheduler in MATRIX. To increase the per client throughput the tasks are submitted in batches. The number of clients is typically configured in a 1:1 ratio between clients and schedulers. The way a client submits tasks can vary according to requirements.

How is this work different than the related work

There is some ongoing work on Falkon which is also a task execution framework but it is a centralized MTC task scheduler if compared with MATRIX. Even the throughput of MATRIX is as high as 13k tasks/sec and efficiency above 90% while Falkon only achieves 20% efficiency. The paper also compares MATRIX with Sparrow and CloudKon in cloud upto 64 EC2 instances and MATRIX outperforms the both by 9X and 5X respectively with 67k tasks/sec and 80% efficiency. Paper ahs discussed the work of SimMatrix is an event simulator a task execution framework at exascale systems. SimMatrix simulates MTC task execution framework comprising of millions of nodes and billions of cores/tasks. SimMatrix supports both centralized (e.g. first-in-first-out or FIFO) and distributed (e.g. work stealing) scheduling. There are many runtime systems like Charm++ , Legion , STAPL , and HPX resource management systems (e.g. SLURM, Slurm++) the I/O forwarding system, IOFSL , which is a scalable, unified I/O forwarding framework for HPC systems; the interconnect fabric managers, OpenSM , which is an InfiniBand subnet manager; and the data aggregation system, MRNet, which is a software overlay network that provides multicast and reduction communications for parallel and distributed tools. These are the types of system software that will be targeted for design explorations with our simulator. There are many fine grained task scheduling systems of HPC which have been under development for many years such as, Falkon , Mesos , YARN , Omega , Sparrow , CloudKon .

YARN and Mesos are two frameworks that decouple the resource management infrastructure from the task scheduler of the programming model to enable efficient resource sharing in general commodity Hadoop clusters for different data-intensive applications. Both apply a centralized resource manager to allocate resources to applications. Omega is a cluster scheduling system that employs specific distributed schedulers for different applications. Like Slurm++, omega shares global resource information to all the schedulers through a centralized master, and the schedulers cache the global information and use an atomic operation to resolve resource conflict on all the compute nodes

Identify the top 3 technical things this paper does well

- The paper comprehensively presented the work and implementation on SimMatrix event simulator.
- The paper has given detailed implementation of MATRIX in MTC applications in exascale system.
- The paper has explained the Adaptive Work Stealing Technique which can be used in exascale systems for load balancing.

Identify 3 things the paper could do better

- Paper has confirmed with respect to the worst-case scenario when MATRIX and Sparrow & CloudKon are tested on Amazon cloud. It is generally difficult to get higher efficiency for shorter jobs since load balancing is not perfect for shorter running tasks.
- The paper could have discussed about comparison between SimMatrix and Falkon, CloudKon to further explain the comparison.
- The work stealing technique, even under a variety of task dependencies, can perform well (83%~95% efficient, depending on the dependency patterns and scale) at scale.

If you were to be an author of a follow up paper to this paper, what extensions would you make to improve on this paper?

I would have used the interposes communication which are transfer from one node to another using shared file system to get high utilization and efficiency on large-scale systems. We can plan to investigate the use of work stealing in improving the performance of GetMTC as accelerators are achieving ever-growing number of cores.