From High-Throughput Computing to Many-Task Computing: Challenges, Systems, and Applications

Jik-Soo Kim, Sangwan Kim*, Seokkyoo Kim, Seoyoung Kim, Seungwoo Rho, Ok-Hwan Byeon, and Soonwook Hwang

National Institute of Supercomputing and Networking,
Korea Institute of Science and Technology Information (KISTI),
Daejeon, Republic of Korea
{jiksoo.kim, sangwan, anemone, sssyyy77, seungwoo0926, ohbyeon, hwang}@ksiti.re.kr

Abstract. Recent emerging applications requiring millions or even billions of tasks (communicating each other through files) to be processed with relatively short per task execution times have expanded the traditional High-Throughput Computing (HTC) and High-Performance Computing (HPC) into Many-Task Computing. Due to some of unique application characteristics, traditional middleware systems that have been widely used in HTC or HPC areas cannot effectively support MTC applications. In this paper, we investigate applications, challenges and systems of MTC and propose key factors of a successful middleware system to fully support MTC applications. We hope our research can give an insight for a next generation distributed middleware system that can support the most challenging scientific applications.

Keywords: Many-Task Computing, Distributed File System, Parallel Processing Framework, Decentralized Data/Compute Management System, Dynamic load balancing

1 Designing a Middleware System for Many-Task Computing

Distributed computing systems have evolved over decades to support various types of scientific applications and overall computing paradigms have been categorized into HTC (High-Throughput Computing) to support bags of (independent) tasks which are usually long running, HPC (High-Performance Computing) for processing tightly-coupled communication-intensive tasks on top of dedicated clusters of workstations or Supercomputers, and Data-intensive Computing leveraging distributed storage systems and parallel processing frameworks. However, Raicu et al. [1] claim that there are many scientific applications that cannot be effectively supported by traditional HTC or HPC systems and proposed a new computing paradigm called *Many-Task Computing* (MTC) to bridge the gap between HTC and HPC.

Recent emerging applications from a wide range of scientific domains (e.g., astronomy, physics, pharmaceuticals, chemistry, etc.) often require a very large number of tasks (from tens of thousands to billions of tasks), and have a large variance of task execution times (from hundreds of milliseconds to hours). Some of

unique application characteristics driving this new computing paradigm are as followings:

- A very *large* number of tasks (i.e. millions or even billions of tasks)
- Relatively *short* per task execution times (i.e. seconds to minutes long)
- *Data* intensive tasks (i.e., tens of MB of I/O per CPU second)
- A large *variance* of task execution times (i.e., ranging from hundreds of milliseconds to hours)
- Communication-intensive, however, not based on message passing interface (such as MPI) but through *files*

For example, stacking of astronomy images involve many tasks ranging from 10K to millions of tasks, each requiring 100ms to seconds of compute and 100KB to MB of input and output data, and also applications such as protein docking require more than one billion computations with a large variance of execution times from seconds to hours.

Relatively short per task running times, a wide variance in task execution times and the communication pattern through files are distinguishing MTC from traditional HTC or HPC. This makes the existing middleware systems such as Condor [2] or BOINC [3] difficult to support MTC applications due to lack of enough resources support, inefficiencies in task dispatching, unreliable and high-latency interconnects.

Condor systems typically employ clusters of workstations in a limited number of organizations (throughout Flocking mechanisms and Condor-G for Grids [2]) which makes it difficult to support millions or even billions of tasks within relatively short periods of time. Also, similar to other local resource managers such as PBS or SGE, Condor has high queuing and dispatching overheads so that it cannot effectively support many short-running tasks [4].

Desktop grid computing systems such as BOINC [3] can integrate enough number of computing resources consisting of personal desktops and workstations over Internet to support processing of many tasks. However, due to the characteristics of wide-area network, it can be challenging to support data-intensive and file-based communicating MTC applications.

Perhaps, existing data-intensive computing systems such as MapReduce [5] combined with HDFS [6] are the most appropriate for MTC applications by storing large amounts of data on scalable and reliable distributed file systems and processing them in parallel by placing computation *close* to the data. However, systems such as Hadoop mainly target batch processing workloads and its use of new programming model based on Map&Reduce can be an obstacle to accommodate existing MTC applications.

Therefore, in order to fully support MTC applications, we need a *system that can efficiently process a very large number of tasks within relatively short periods of time, effectively support data-intensive and file-based communicating patterns, and adapt to dynamically changing load distribution due to a large variance in task execution times.* To summarize, the goals of middleware systems for MTC applications must include the following:

- Efficient task dispatching mechanisms that can expedite processing a very large number of tasks
- Ability to harness as many computing resources as possible to support multiple users submitting large numbers of tasks
- Intelligent scheduling mechanisms that can automatically select more responsive and effective resources
- Minimizing user overhead of handling a large amount of jobs and computing resources
- Mechanisms for minimizing data movements to improve the performance of data-intensive and file-based communicating MTC applications
- Dynamic load balancing that can adjust acquired resources according to changing load distribution due to heterogeneity in task execution times and computing resource capabilities

Multi-level scheduling techniques, where a first-level scheduler reserves resources by submitting pilot jobs to batch schedulers, and then lightweight second-level scheduler dispatches jobs directly to the reserved resource pool, can be employed to circumvent the performance bottleneck of traditional batch schedulers (e.g., PBS, SGE, Condor). This mechanism effectively creates a dedicated resource pool on the fly for fast dispatching of many tasks through bypassing local batch schedulers and can be applied to clusters of workstations or even Supercomputers [7]. Because of the rich functionalities of batch schedulers, submitting millions or even billions of tasks can cause significant overheads and result in high dispatching time per task [1, 4]. By employing effective multi-level scheduling techniques, some of middleware systems such as Falkon [4, 7] or MyCluster [8] could achieve more than 50% of performance improvements over traditional single-level batch schedulers. However, ensuring *fairness* among multiple users submitting various numbers of tasks independently to the collection of computing resources is still an open research problem.

To support complex and demanding scientific applications consisting of many tasks, it is inevitable to harness as many computing resources as possible including Supercomputers, Grids, and even Cloud. However, it is challenging for researchers to utilize available resources that are under control by independent resource providers as the number of jobs (that should be submitted *at once*) increases dramatically (as in parameter sweeps or N-body calculations). Therefore, we need mechanisms and systems that can integrate and manage vast amounts of heterogeneous computing resources, intelligently select most appropriate computing resources for different types of users and applications, and provide easy-to-use tools for submitting a very large number of tasks at once.

Finally, a wide variance in task execution times and data-intensive characteristics of MTC applications still leave a lot of research issues although some of dynamic resource provisioning mechanisms [4, 8] and data-aware scheduling & caching mechanisms [9] are proposed. This is because as the number of tasks increases, it becomes more difficult to support data-intensive computing applications with heterogeneous execution times due to lack of high-performance shared storage support and increased complexity of managing and dynamically and globally adjusting many tasks. Current shared file systems that are common in petascale systems cannot effectively support data-intensive applications so that we may need a

complete new storage architecture that can partition data across multiple nodes and leverage local disks to form a distributed file system [10].

2 Conclusion

Building an efficient and effective system for MTC applications that often require a very large number of tasks, and have a large variance of task execution times and data-intensive computing/communication are still challenging and leaving us a lot of research issues. However, we believe that leveraging distributed file systems, parallel processing frameworks, decentralized data/compute management systems, dynamic load balancing techniques will be crucial to design and implement a next generation middleware system that can support the most challenging scientific applications.

References

1. Raicu, I., Foster, I., Zhao, Y.: Many-Task Computing for Grids and Supercomputers. In: Proceedings of the Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS'08). (November 2008)
2. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency and Computation: Practice and Experience 17(2-4) (2005) 323–356
3. Anderson, D.: BOINC: A System for Public-Resource Computing and Storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004). (November 2004)
4. Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I., Wilde, M.: Falkon: a Fast and Light-weight tasK executiON framework. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC'07). (November 2007)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings of the USENIX 6th Symposium on Operating Systems Design and Implementation (OSDI'04). (May 2004)
6. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10). (May 2010)
7. Raicu, I., Zhang, Z., Wilde, M., Foster, I., Beckman, P., Iskra, K., Clifford, B.: Towards Loosely-Coupled Programming on Petascale Systems. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC'08). (November 2008)
8. Walker, E., Gardner, J.P., Litvin, V., Turner, E.L.: Creating Personal Adaptive Clusters for Managing Scientific Jobs in a Distributed Computing Environment. In: Proceedings of the Challenges of Large Applications in Distributed Environments (CLADE'06). (June 2006)
9. Ioan Raicu, Yong Zhao, I.F., Szalay, A.: Accelerating Large-Scale Data Exploration through Data Diffusion. In: Proceedings of the 2008 ACM International workshop on Data-aware distributed computing (DADC'08). (June 2008)
10. Raicu, I., Foster, I.T., Beckman, P.: Making a Case for Distributed File Systems at Exascale. In: Proceedings of the ACM Workshop on Large-scale System and Application Performance (LSAP'11). (June 2011)