Towards a Next Generation Distributed Middleware System for Many-Task Computing

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

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 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. Many-Task Computing (MTC) aims to bridge the gap between traditional HTC and HPC by building efficient middleware systems throughout employing lightweight task dispatching mechanisms, minimizing data movements, leveraging data-aware scheduling, and proposing of next generation Exascale storage systems. Recent emerging applications requiring millions or even billions of tasks to be processed with relatively short per task execution times have driven this new computing paradigm.

In this paper, we investigate concepts and technologies of MTC and propose guidelines for building an efficient and effective middleware system to fully support MTC applications. Throughout our short survey about challenges, systems and applications of MTC, we argue that a next generation distributed middleware system must effectively leverage distributed file systems, parallel processing frameworks, decentralized data/compute management systems, and dynamic load balancing techniques to solve the most challenging and complex scientific problems.

Keywords: High-Throughput Computing, High-Performance Computing, Many-Task Computing, Data-intensive Computing, Middleware System

1. Introduction

Distributed computing systems have evolved over decades to support various types of scientific applications ranging from compute-intensive to data-intensive, from independent to dependent, and from long-running to short-running. Dependency among multiple tasks of scientific applications also can be divided into two different categories: communication dependency (whether there is a need for communication between tasks) and execution flow dependency (whether task executions are pipelined and form a workflow). To support
communication-intensive and tightly-coupled parallel computations, message passing interfaces such as MPI [1] or PVM [2] have been developed and widely used especially in dedicated clusters of workstations or petascale systems such as IBM’s Blue Gene/P [3] (often called High-Performance Computing). Some of research work such as [44] proposed implicit parallelism as opposed to explicitly utilizing MPI or PVM so that the implementation of the parallelism will be automatically realized in the model to support complex HPC applications. Also, several research efforts [4–6] to incorporate another type of dependency among tasks (i.e., workflow) have investigated building effective scientific workflow management systems which can take high-level descriptions of complex applications structured as workflows and automatically map them to available computing resources. Therefore, overall scientific applications can be divided into multiple different categories based on processing requirements (compute or data), communication dependency, execution flow dependency, and task execution times (short running or relatively long running). According to the study of Iosup et al., [7, 8], a high percentage of Grid [9] applications employ an *embarrassingly parallel* model (sometimes called *bags of tasks* workloads) where the majority of tasks are independent (in terms of communication and execution flow).

Computing paradigms such as High-Throughput Computing (HTC) [10] or Volunteer Computing [11] mainly target compute-intensive (relatively low I/O requirements), independent (there is no communication needed between tasks) and long-running applications consisting of many loosely-coupled tasks. Middleware systems such as Condor [12, 13], BOINC [14], and SETI@Home [15] have successfully achieved a tremendous computing power by harnessing a large number of computing resources consisting of either clusters of workstations or desktop machines over Internet. In addition to traditional centralized Grid architectures, peer-to-peer (P2P) desktop grid computing systems have been proposed to circumvent the performance bottlenecks and limited scalability by incorporating decentralized P2P techniques [16, 17]. The overall system, from the point of view of a user, can be thought of as a combination of a centralized, Condor-like grid system for submitting and running arbitrary jobs, and a system such as BOINC or SETI@Home for farming out jobs from a server to be run on a (potentially very large) collection of machines in a completely distributed environment. However, since the problem of finding an optimal mapping of tasks onto the computing resources still has been shown, in general, to be NP-Complete [45], many on-line scheduling algorithms and heuristics are studied. All of these systems aim to support embarrassingly parallel and compute-intensive workloads by integrating vast amounts of computing resources.

On the other hand, within many science domains, the data generally grows faster than computational resources and their speed [18] so that it becomes crucial to effectively store and manage very large data sets and efficiently process them in parallel by minimizing data movements. Distributed file systems such as HDFS [19], Google File System [20] or Sector [21] can store terabytes or even petabytes of data reliably and scalably by efficiently partitioning data and storing them redundantly across (thousands) of hosts based on commodity hardware. To minimize data movements, parallel processing frameworks such as MapReduce [22] or Sphere [21] can execute application computations in parallel *close* to their data. One of the main advantages of these distributed file systems is that a cluster can scale computation capacity, storage capacity and IO bandwidth by simply adding commodity servers which can greatly reduce the overall cost of IT resource management. This enables Hadoop to become a standard platform in “Big Data” and it has been widely adopted not only in research community but also in commercial fields (*e.g.*, Yahoo!, Facebook, Cloudera, IBM, Dell, Oracle, EMC, Teradata, Microsoft, Microstrategy, etc.).
Therefore, overall computing paradigms have been categorized into HTC (including Volunteer Computing) to support bags of tasks which are usually long running, HPC for processing communication-intensive and tightly-coupled parallel tasks on top of dedicated clusters of workstations or Supercomputers, and Data-intensive Computing based on distributed storage systems and parallel processing frameworks. However, Raicu et al., [18] 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.

In this paper, we investigate concepts and technologies of MTC and propose guidelines for building an efficient and effective middleware system to fully support MTC applications. Throughout our short survey about challenges, systems and applications of MTC, we aim to give an insight into research community to design and implement a next generation distributed middleware system. The rest of our paper is structured as follows. Section 2 describes the definition, challenges and applications of MTC. Then, we will propose guidelines for building an effective middleware system that can fully support MTC applications and describe some of existing technologies that can be adopted. We conclude our paper at Section 3.

2. MTC: Challenges, Systems and Applications

Recent emerging applications requiring millions or even billions of tasks to be processed with relatively short per task execution times have led the traditional HTC and HPC to expand into MTC [18]. These 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), and have a large variance of task execution times (from hundreds of milliseconds to hours). Some of key 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

![Stacking of astronomy images](image-url)
For example, stacking of astronomy images (Figure 1a) involves 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. Applications such as Figure 1c require more than one billion computations with a large variance of execution times from seconds to hours (average 10 minutes). If we skim over the characteristics and definitions of MTC, it is not clear to identify the major differences between MTC and HTC, HPC or Data-intensive Computing. As we mentioned in Section 1, HTC mainly targets compute-intensive and independent (in terms of communication) loosely-coupled long running applications so that primary metrics are the number of operations processed per month [10]. On the other hand, in HPC area, many applications are required to process tightly-coupled parallel computations communicating each other through message passing interfaces.

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 or BOINC 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 [12]) 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 [23] or SGE [24], Condor has high queuing and dispatching overheads so that it cannot effectively support many short running tasks [25, 26]. Desktop grid computing systems such as BOINC or SETI@Home 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 and opportunistic sharing of volunteered resources, it can be challenging to support data-intensive and file-based communicating MTC applications consisting of many tasks.
Perhaps existing data-intensive computing systems such as MapReduce combined with HDFS 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 usually target batch processing workloads and its use of new programming model based on map & reduce can be an obstacle to support existing “black box” 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 for 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
- Ability to support black box application execution as is commonly found in MTC or HTC applications

Multi-level scheduling techniques [27] 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 [26]. For example, as we can see from the Figure 2, the dispatcher in the Falkon [25, 26] efficiently dispatches tasks to the executors on the computing resources which can be dynamically acquired by the provisioner. In the Falkon system, the GRAM4 [42] is used as the first-level scheduler to reserve the computing resources. 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 [18, 25]. Some of middleware systems such as Falkon or MyCluster [28] 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.

![Diagram of Falkon Multi-level Scheduling System](image)

**Figure 2. The Architecture of Falkon Multi-level Scheduling System [25]**

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. Especially, cloud computing is shifting the IT paradigm from owning to borrowing by providing mechanisms for instant provisioning of compute/storage resources on demand and affects not only to overall IT industries but also to governmental systems [43]. 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*) increase 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. Utilizing pluggable resource interfaces to integrate heterogeneous computing resources, providing a method that can easily describe many computations in a single job script such as OGF JSDL Parameter Sweep Extension [46] can be employed to address these issues. Although some of researches integrate Cloud resources such as Amazon EC2 [29] to support many-task computing [30–32], they cannot exploit different types of computing resources intelligently yet and mainly focus on applying Hadoop MapReduce processing on the Cloud.

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 [25, 28, 33] and data-aware scheduling & caching mechanisms [34] 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 globally managing and dynamically adjusting many tasks.

Dynamic load balancing is crucial to tackle the heterogeneity in task execution times and resource capabilities. Even if the initial load balancing mechanism assigns the jobs uniformly across available system resources, as time passes the overall load distribution may change because some nodes run the allocated jobs much faster than others (or some jobs just have relatively short running times). Therefore, the overall throughput of the entire system may heavily depend on its slowest nodes. Also, it can be very difficult in general to predict the actual running time of a job on a given node in advance, unless clients provide such information and it is accurate for all node types in the system. However, the actual queuing
time for a job is not necessarily directly proportional to the number of jobs in the queue, since the job running times can vary widely. Therefore, we need a mechanism that can consistently monitor the overall load distribution changes and effectively adapt to the changing load. This can be done in various ways, either redistributing the tasks from one node to another, or dynamically adjusting the number of acquired resources per user.

To support data-intensive MTC applications, the ability to distributed data over hundreds of computers and achieve data locality becomes crucial as we could see from Hadoop HDFS and MapReduce frameworks. Raicu et al., [34] proposed an alternative approach to accomplish data locality through data diffusion approach which leverages local resource caches and data-aware scheduling in Falkon (as we can see from the Figure 3). This approach intends to move the data on demand to the computing resources (employing data caching mechanism to reduce the data transfer cost) rather than dedicate computing and storage resources to analysis tasks (as in Hadoop). Falkon implements this approach by adding the data-aware scheduler in the dispatcher component and each executor can utilize the data cached nearby neighbors. In the case of cache miss, executors always can directly access the persistent storage to get the required data and it is the responsibility of the dispatcher to manage information about all of cached data and inform the executors. However, as in other caching-based data-intensive computing research, data diffusion mechanism is heavily dependent on the characteristics of application data locality (i.e., whether there are enough data locality in the input data) so that the performance improvements can vary widely depending on applications.

![Figure 3. Leveraging Data Diffusion & Data-aware Scheduling in Falkon [34]](image)

Performance challenges for the processing of large data sets are also in the Supercomputers, as current shared file systems [35–37] that are common in petascale systems cannot effectively support data-intensive applications as we can see from [38]. Therefore, we may need a complete new storage architecture that can partition data into multiple nodes and leverage local disks to form a distributed file system [38, 39]. Raicu et al., [38] proposed to exploit nonvolatile memory (e.g., solid state memory) on every compute node (as we can see from Figure 4) and connect them with the many orders of magnitude higher bisection bandwidth in multi-dimensional torus networks. In this scheme, every compute node actively participates in the metadata and data management which can result in leveraging the abundance of computational power of many-core processors. This means that the philosophy and the data processing model of Hadoop can be also applied to the petascale systems which can guide us to design the next generation Exascale computing systems.
To manage a very large number of tasks and dynamically adjust them according to changing load distribution, some of decentralized technologies [40, 41] can be applied to build more scalable and robust systems. This will remove a single point of failure and contention as commonly found in the centralized batch processing framework such as Hadoop or Condor where the centralized server is responsible for managing all the metadata (or job executions) which can limit the reliability and scalability of the system.

3. Conclusion

As we described in this paper, Many-Task Computing was proposed to address some unique challenges from various scientific applications which cannot effectively supported by existing HTC or HPC technologies. However, while we were investigating the concepts and technologies of MTC, we realized that the problems are not restricted only to the MTC area but it can also give us important lessons to improve the current compute/data processing techniques, designing a next generation distributed middleware system, and Exascale computing systems.

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. With upcoming era of Exascale Computing, 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


