J Internet Serv Appl (2012) 3:77–86
DOI 10.1007/s13174-011-0054-7

SI: FOME - THE FUTURE OF MIDDLEWARE

On Cloud computational models and the heterogeneity challenge

Raouf Boutaba · Lu Cheng · Qi Zhang

Received: 28 November 2011 / Accepted: 29 November 2011 / Published online: 9 December 2011

Abstract Cloud computing is by far the most cost-effective technology for hosting Internet-scale services and applications. The MapReduce model, in particular, is largely used nowadays in Cloud infrastructures to meet the demand of large-scale data and computation intensive applications. Despite its success, the implications of MapReduce on the management of Cloud workload and cluster resources are still largely unstudied. In this article, we show that dealing with the heterogeneity of workloads and machine capabilities is a key challenge. In today's cloud environment, workloads can have varied sizes, lengths, resource requirements, and arrival rates. The machines also have varied CPU, memory, I/O speed, and network bandwidth capacities. Jointly they pose difficult challenges pertaining, among others, to job scheduling, task and data placement, resource sharing and resource allocation. We analyze the heterogeneity challenge in these specific problem domains and survey the representative state-of-the-art works that try to address them. We found that although advances are made that partially address some of the outlined challenges, there are even more open challenges yet to be explored, and this topic at large is ripe for scientific contributions.

R. Boutaba (✉) · L. Cheng · Q. Zhang
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
e-mail: rboutaba@uwaterloo.ca

L. Cheng
e-mail: l32cheng@uwaterloo.ca

Q. Zhang
e-mail: q8zhang@uwaterloo.ca

R. Boutaba
Division of IT Convergence and Engineering, POSTECH, Pohang 790-784, Korea

Keywords Cloud computing · MapReduce · Heterogeneity · Scheduling · Resource allocation

1 Introduction

Cloud computing has become the most cost-effective technology for hosting Internet-scale applications. Companies like Google and Facebook generate enormous volumes of data on a daily basis that need to be processed in a timely manner. To meet this requirement, Cloud providers use computational models such as MapReduce [9] and Dryad [17]. In these models, a job spawns many small tasks that can be executed concurrently on multiple machines, resulting in significant reduction in job completion time. Furthermore, to cope with software and hardware exceptions frequent in large-scale clusters, these models provide built-in fault tolerance features that automatically restart failed tasks when exceptions occur. As a result, these computational models are very attractive not only for running data-intensive jobs, but also for computation-intensive applications.

The MapReduce model, in particular, is largely used nowadays in Cloud infrastructures for supporting a wide range of applications and has been adapted to several computing and cluster environments. Despite this success, the adoption of MapReduce has implications on the management of Cloud workload and cluster resources, which is still largely unstudied. In particular, many challenges pertaining to MapReduce job scheduling, task and data placement, resource allocation, and sharing are yet to be addressed.

Several studies attempted to characterize the workload in production MapReduce clusters (e.g., [5, 6, 18, 21, 28]). An analysis of the data reported in these studies reveals a key observation about the *heterogeneity* of the workload in terms of job sizes, lengths, resource requiremens and arrival rate.

The difference in job sizes can span several orders of magnitude. Although many jobs contain only a few tasks, a large job can spawn thousands of parallel tasks to speed up its execution, consuming a lot of computation resources in a short period. The length of jobs can also differ significantly. Although most jobs have short running time, some jobs can take a very long time (e.g., several days) to complete. Jobs also have heterogeneous resource demands. Some jobs are CPU intensive, some are memory intensive, and some have high demand for I/O speed or network bandwidth. For instance, Mishra et al. [21] reported that Google's compute clusters are often shared by application tasks with diverse service level requirements in terms of throughput and latency. Finally, like many other service systems, the arrival rate of job requests can be spiky. In order to ensure acceptable response time during peak workload time, the capacity of a cluster is much higher than what an average workload needs, resulting in low resource utilization during off-peak periods. To address this issue, a common practice is to share the cluster resources by mixing jobs with different priorities. Typically, production jobs (i.e., jobs that generate revenue) are given higher priorities than nonproduction jobs (e.g., research experiments). As a result, although production jobs account for a small percentage of the total job population, they are allowed to consume a significant portion of the cluster resources.

In addition to workload heterogeneity, the machines in MapReduce clusters are heterogeneous in resource capacities and performance capabilities. This is because Cloud providers typically leverage computing resources purchased from previous investments, resulting in multiple generations of server and networking equipment within the same cluster.

The aforementioned heterogeneity of workloads and machines introduces significant complexity, which needs to be effectively dealt with in order to achieve efficient MapReduce computations and cluster management. More specifically, in job scheduling, heterogeneous job characteristics and requirements affect efficiency and fairness of job completions; in data and task placement, heterogeneity hinders job completion rate and increases communication overhead; in resource sharing, heterogeneous job requirements calls for flexible and job-specific capacity allocation to achieve optimal machine utilization; in performance-aware resource allocation, performance models need to be developed in order to ensure the performance objectives of each job are achieved. There has been a large body of recent work on improving the performance of MapReduce computations. As MapReduce applications are becoming critical to the operations of cloud companies, optimizing their performance is becoming a hot research topic and has attracted many researchers from different fields.

The contributions and organization of this paper are as follows. We first provide an overview of the MapReduce computational model and describe Apache Hadoop, the most popular open-source implementation of MapReduce (Sect. 2). Second, we provide a detailed analysis of workload characteristics in production clusters at Microsoft, Google, and Facebook (Sect. 3). Our analysis particularly emphasizes the heterogeneity of workloads and machines in Cloud computing environments. Third, we discuss the key research challenges posed by heterogeneity in production MapReduce clusters and highlight related research directions (Sect. 4). Finally, we survey in Sect. 5 the state-of-the-art work trying to address some of these key research challenges and draw our conclusions in Sect. 6.

2 Cloud computational models

Cloud computing is defined by the National Institute of Standards and Technology (NIST) as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [22]. Essentially, Cloud computing is not a new technology, but rather a new operational model that uses existing technologies to meet the technological and economical requirements of today's demand for high performance, flexible, scalable, and cost-effective IT infrastructure. Cloud computing is similar to Grid computing in that it also employs distributed resources to achieve application-level objectives. However, cloud computing takes one step further by leveraging virtualization technologies at multiple levels (hardware and application platform) to realize resource sharing and dynamic resource provisioning. Cloud computing adopts a utility-based pricing scheme entirely for economic reasons. With on-demand resource provisioning and utility-based pricing, service providers can truly maximize resource utilization and minimize their operating costs. Virtualization is fundamental to Cloud computing, as it provides the capability of pooling computing resources from clusters of servers and dynamically assigning or reassigning virtual resources to applications on-demand. For a description of Cloud computing concepts, architectural principles, state-of-the-art implementations and research challenges, the reader is referred to [29]. The focus of this paper is on Cloud computational models designed for large scale computations commonly used in Cloud environments today. In particular, this section introduces MapReduce and its most popular open-source implementation, Apache Hadoop.

2.1 The MapReduce computing model

Parallel computing frameworks like MapReduce [9] and Dryad [17] have become the dominant programming models for data-intensive computing in Cloud computing environments. Originally proposed by Google, MapReduce has

been designed for processing and generating large data sets [9]. In essence, MapReduce exploits the well-known divide and conquer design philosophy. A MapReduce job consists of two types of tasks, namely map and reduce tasks. The original input of a MapReduce job is divided into multiple file blocks of equal size, typically 64MB or 128MB. Each file block is processed by a map task that generates a set of intermediate key/value pairs. Map tasks are independent of each other and do not communicate or exchange data during execution. Reduce tasks are responsible for merging all the intermediate values associated with the same key and generate the final output. For example, consider a word count job whose input may consist of a large collection of files. A single map task emits each word plus an associated count of occurrences in the particular file block it processes. These map tasks are executed independently of each other. In the next phase, intermediate results produced by map tasks will be shuffled, sorted, and it is the responsibility of reduce tasks to compute the total count for each word and generate the final output.

The MapReduce computing model has two significant benefits. First, it is scalable. Map tasks as well as reduce tasks can be distributed and executed in parallel on multiple machines. A large job with hundreds of gigabytes input files will spawn thousands of tasks that may be distributed across the whole cluster and executed at the same time to significantly cut down the job completion time. Second, this model is fault tolerant which is of vital importance since software and hardware exceptions are common in large-scale clusters. Because tasks are independent within each phase (map or reduce), every single task can be killed and restarted independently when exceptions happen, without having to re-execute any of the other tasks. Moreover, job schedulers speculatively re-execute tasks that appear to be slower than other tasks of a particular job to further cut down on job response time. These reasons make the MapReduce model ideal for running large-scale data- and computation- intensive applications in Cloud environments.

2.2 Apache Hadoop

Apache Hadoop [4] is a framework that allows for the distributed processing of large data sets across clusters of computers. As a subproject of Hadoop, Apache Hadoop is an open source implementation of the MapReduce model proposed by Google. It is now widely used in many Cloud environments including those of Yahoo!, Facebook, and twitter.

Apache Hadoop uses a master-slave architecture. The master consists of a JobTracker and a NameNode, which usually run on dedicated machines in a large cluster. A Hadoop cluster typically has multiple slaves, each of which acts as both a DataNode and TaskTracker. When a file is copied into Hadoop clusters, the Hadoop Distributed File System (HDFS) [14] splits the file into multiple file blocks of equal size, and stores these file blocks on DataNodes. HDFS uses NameNode to locate a file block.

Before users submit their jobs, input files of jobs must have already been copied into the cluster's filesystem. Once the data becomes available, users can submit jobs to the Job Tracker, and the Job Tracker uses job schedulers to assign tasks spawned by the job to TaskTrackers that actually execute these tasks. The data block of a job may or may not reside on a TaskTracker that executes the tasks of that particular job. In the latter case, data must be transferred to that TaskTracker before the tasks can proceed. TaskTrackers report the status of tasks to the JobTracker in heartbeat messages every few seconds, and the Job Tracker uses this information to make scheduling decisions, speculatively re-execute tasks, and restart failed tasks in case of exceptions.

3 The heterogeneity challenge

Cloud computational models must consider many factors such as energy consumption, resource utilization, job response time, performance variability and cost. These factors often conflict with one another, and as a result tradeoffs are often required. The impact of these factors is further exacerbated by the heterogeneity of workloads and resources in Cloud computing environments. Understanding the impact of heterogeneity is fundamental for devising such complex tradeoffs, and consequently achieving better performance in a cost-effective manner.

Recent literature reported useful data on typical workloads in production clusters at Microsoft, Google, and Facebook. Based on these data, we provide in this section a detailed analysis of the heterogeneity challenge faced by Cloud computational models in terms job lengths, job sizes, arrival rates, performance requirements, and machine capabilities.

3.1 Bimodal distribution of job lengths

Table 1 shows the distribution of MapReduce and Dryad job response time. The data are collected from a dedicated 600-node Hadoop cluster at Facebook [28], a 40-node Hadoop at an anonymous Internet Company (hereon referred to as IC) [5], and a production cluster used inside Microsoft's search division [18].

Table 1(A) shows that at Facebook the median job length is 84 s and the average length of jobs is between 300 s and 450 s. Table 1(B) indicates that the median as well as average job response times at IC are close to those at Facebook. In the Microsoft's research cluster, the median job is approximately 30 minutes, as shown in Table 1(C). Average job lengths at Facebook and IC are close to Google's statistic of 395 s [21], while significantly shorter than the average length of jobs in Microsoft's research cluster.

Table 1 CDF of MapReduce/Dryad jobs response times at Facebook, Internet Company, and Microsoft

(A): Data in a Hadoop cluster at Facebook

%Jobs	40%	50%	60%	70%	80%
Job Run time (s)	55	90	120	250	350
%Jobs	90%	95%	98%	99%	99.5%
Job Run time (s)	650	1200	3000	5000	>10000

(B): Data in a Hadoop cluster at Internet Company

%Jobs	40%	50%	60%	70%	80%	90%
Job Run time (s)	45	80	130	190	450	650

(C): Data in a Microsoft research cluster

%Jobs	18.9%	28.0%	34.7%	51.3%	72.0%	95.7%
Run time (minutes)	5	10	15	30	60	300

Table 2 CDF of number of map tasks In a Hadoop cluster at Facebook

% Jobs	39%	55%	69%	78%	84%	90%
#Maps	1	2	20	60	150	300
% Jobs	94%	97%	98%	99%	99.5%	The largest in a week
#Maps	500	1500	3065	3846	6232	25000

From Table 1, it is evident that most jobs in current Cloud computing clusters are short. However, in all of the three clusters there are long jobs with running time at least 2 to 3 orders of magnitude longer than the running time of the short ones. Moreover, the distributions of job response time show bimodal behavior, with a transition region between 200 and 400 seconds at Facebook and IC, and between 60 and 100 minutes at Microsoft.

More recently, similar observations have been reported by Mishra et al. [21] on workload characteristics in Google's cloud backend. Even though their dataset contains non-MapReduce jobs, the workload characteristics reported in [21] are still similar to what we can see in Table 1. Specifically, the duration of task executions follows a bimodal distribution as tasks either have short or long running time. Even though most tasks are short, long tasks are multiple orders of magnitude longer than the short ones.

3.2 Bimodal distribution of jobs sizes

How about the distribution of job sizes? Table 2 describes the job size distribution in terms of number of map and reduces tasks at Facebook [28]. The number of map tasks

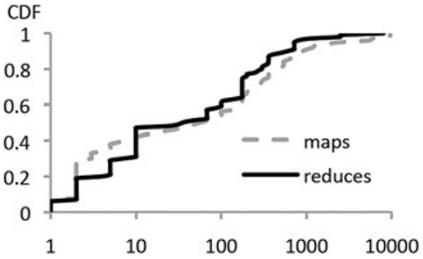

Fig. 1 CDF of the number of map and reduce tasks in a Hadoop cluster at Internet Company [5]

reflects the input file size, because in Hadoop clusters, the number of map tasks is equal to the input file size divided by the size of a block, which is normally 64MB or 128MB. From Table 2, it can be observed that most jobs have a small number of map tasks. On the other hand, a small fraction of large jobs have very large number of map tasks. In the experimental workload used in [28], which is designed based on the metrics of real workloads at Facebook, among 100 jobs the 4 largest jobs account for 73% of the total 26,410 map tasks. Figure 1 shows the number of map and reduce tasks per job at IC. Around 40% jobs contain less than 10 map or reduce tasks, but another 40% jobs or so have more than 100 map or reduce tasks. Also, at Facebook and IC, the smallest jobs each contains only one task, but a single large job can have up to tens of thousands of tasks. Similar to the length of jobs, the distribution of job sizes is bimodal [5].

Based on the above discussion, we know the job length and size both can vary across multiple orders of magnitude. Statistically, if we add the running time of all the jobs together, a few long jobs constitute a large portion of the total running time; and if we add up the number of all the tasks, a few large jobs take a large proportion of the total number of tasks. These characteristics also match with have been observed at Google [21] where most resources are consumed by a few jobs with long duration that have large demands for CPU and memory.

The aforementioned characteristics of MapReduce workloads share many similarities with what is already known in traditional distributed computing systems [10]. Harchol-Balter [16] observed workload in many real environments and found that it has a mixture of job lengths and sizes spanning many orders of magnitude. Typically, there are many small jobs and a few large ones. As another example, measurements have shown that running time of Unix processes, sizes of files transferred through the Web and stored in Unix file systems have heavy-tailed distributions [11, 15, 16].

3.3 Fluctuating job arrival rates

Similar to job sizes and length, the arrival rate of MapReduce jobs is also highly variable from time to time. The distribution of job interarrival times at Facebook in October

2009 was first reported in [28]. Chen et al. [5] also studied job interarrival times at Facebook and IC. For both companies, interarrival time exhibits an on-off pattern according to the time of the day. During daytime, job arrival can be intense, with around 40% of job interarrival times less than 10 s. Consequently, the system is often very busy. However, at nighttime, job arrival intervals can be very long. In this case, most resources in the clusters become idle.

Combined with job characteristics discussed above, we can further see that slot requests in clusters are even spikier than job arrival rates. This is because the distribution of job sizes is bimodal and, therefore, a large portion of the requests for computation slots will arrive at the same time with the arrival of large-sized jobs.

Bursty job arrival rate is also commonly observed in other service systems like telecommunication networks and public transportation systems. For example, in VoIP networks, the traffic during the busiest hour accounts for approximately 15 to 20% of the traffic for that day [26]. Similar measurements have been reported for Telecommunications networks [19, 20]. In these systems, the average traffic intensity during the busiest periods is 3 to 4 times higher than the average workload intensity.

3.4 Heterogeneous resource requirements of jobs

In addition to the size and the length, jobs have very heterogeneous resource demands. Some jobs are CPU intensive, some are memory intensive, and some have high demand for I/O speed and network bandwidth. This diversity often intensifies resource contention or resource wastage. For example, when multiple CPU-intensive tasks are scheduled on the same machine, the contention for CPU resource among the tasks will cause them to slow down. At the same time, however, there may be a large portion of unused memory wasted on the same machine. Even if the amounts of unused resources on a single machine are small, they accumulate to significant resource wastage in a large cluster. In addition to contention for CPU, memory and networking resources, MapReduce tasks also compete for locations where they are executed. Many jobs have data blocks distributed across the entire cluster. Constrained by network bandwidth, processing a local data block is faster than copying data from a remote machine and then processing it. During the reduce phase, a single reduce task will communicate with many map tasks, which may generate large volume of traffic and cause network delay. Therefore, achieving data locality can significantly improve the performance of a data intensive task. All these contentions can cause a common phenomenon called *stragglers*, which refers to tasks significantly lagging behind other tasks of a particular job. Therefore, it is a key challenge to mitigate resource contention while maintaining high resource utilization in the cluster.

3.5 Heterogeneous hardware

Machines in MapReduce clusters can also have heterogeneous capabilities. The reason is that in order to make use of previous investment, there are often multiple generations of server and networking equipments in a cluster [27]. Such a heterogeneous execution environment has great impact on resource sharing among jobs. In the original MapReduce paper, Dean et al. [9] noted that speculative execution can improve job response time by 44%, that is, job schedulers speculatively reexecute tasks that appear to be stragglers. This technique was implemented in Apache Hadoop to improve job response time. However, Zaharia et al. [27] observed that the Hadoop job scheduler implicitly assumes cluster machines are homogeneous and tasks make progress linearly, and decides when to speculatively reexecute tasks that appear to be stragglers based on these assumptions. With heterogeneous clusters, however, Hadoop's scheduler can cause severe performance degradation as some servers may have better computation power whereas others can have better storage capacity. It is therefore essential to leverage the heterogeneous capabilities of individual machines to best satisfy resource requirements of individual tasks.

4 Research challenges posed by heterogeneity

The heterogeneity of workload and machines introduces many challenges for effectively managing MapReduce clusters in production Clouds. In this section, we discuss the key challenges posed by heterogeneity, namely: job scheduling, data and task placement, resource sharing, and resource allocation.

4.1 Job scheduling

The responsibility of the job scheduler is to assign tasks to machines with consideration for both efficiency and fairness. To achieve efficiency, job schedulers must reduce resource wastage and maintain high utilization in the cluster. At the same time, fairness is also an important concern since a cluster is often shared by many jobs from multiple users. The most common requirements with respect to fairness are [18]: (1) preventing one user's jobs from monopolizing the whole cluster, delaying the completion of everyone else's jobs; (2) ensuring low latency for small or short jobs while maintaining a high overall throughput. The level of fairness also reflects stability of service quality, as job response time becomes steady and predictable under fair scheduling. This is because when traffic intensity is high in the cluster, fair scheduling will ensure jobs of the same priority will experience similar delays so that no job will be delayed much more heavily than others. Similarly, when workload is light, each

job will experience similar increase in execution rate. A system with reasonable and predictable response time may be considered more desirable than a system that is faster on the average but is highly variable in job performance.

In today's production, clusters up to millions of tasks with different performance objectives such as response time or throughput are executed daily. The conjunction of such scale and the heterogeneity of workloads and machines makes the design of effective scheduling policies very challenging. To a certain extent, job scheduling can be seen as a multi-dimensional bin-packing problem. Even if all tasks have similar characteristics and all the cluster nodes have the same capacity, it is still a well-known combinatorial NP-hard problem. Furthermore, the heterogeneity in hardware, the communication constraints as well as the dynamicity in job arrival rates make this problem even harder to solve in practice.

As mentioned in Sects. 3.1 and 3.2, job sizes and lengths are not evenly distributed. Given that production and non-production jobs often share the same cluster in practice, preemption is commonly used to make sure high priority tasks are given adequate resources and large jobs will not starve others. However, preemption can sometimes significantly delay the completion time of long running jobs and results in a waste of resources. Indeed, the completion of a job requires fulfillment of map tasks followed by reduce tasks. This feature of the MapReduce programming model results in situations where the whole job is held up in the system even if a single task is lagging behind (possibly due to repeated preemptions), leading in turn to increased response time and wastage of resources.

4.2 Data and task placement

Another key issue somewhat related to MapReduce scheduling is the placement of data and tasks. Modern MapReduce clusters operate on top of distributed file systems (e.g., Google File System [12] and HDFS) where datasets are divided into data blocks of fixed size and replicated on multiple machines. To reduce communication overhead and improve job completion time, it is desirable to achieve data locality by executing a task locally to the input data, or as close to the input data as possible. This raises the question of finding effective placement of data and tasks in MapReduce clusters. It should be pointed out that data placement and task placement are related problems that must be solved jointly, as task placement is often dependent on data placement. Both problems are difficult to solve, as they generalize the bin packing problem. Finally, the communication pattern between data and tasks (e.g., multicasting, shuffling, and incasting) must be taken into consideration when solving these problems.

4.3 Resource sharing

In addition to deciding the placement of tasks on machines, another challenge is to decide how physical resources in the cluster, including CPU, memory, disk I/O, and network resources, should be shared among tasks in each job. As tasks have heterogeneous resource requirements, it is important to decide how much resources should be allocated to each task in order to mitigate performance bottlenecks and to achieve high performance in terms of throughput. For computing resources such as CPU, memory, and disk, Hadoop uses a simple approach that divides the physical resources on each machine into identical slots, and assigns each task to a single slot. Even though this approach achieves fairness, it is often suboptimal for tasks with different resource requirements. For instance, consider a CPU intensive task colocated with a disk intensive task. In this case, it is desirable to allocate more CPU resources to the CPU intensive task, and more disk resources to the disk intensive task. Such an objective is not achievable with the current slot-based allocation scheme. Thus, more effective resource sharing strategies are needed.

Another related issue is the sharing of network resources. Data-intensive tasks often require transferring large volumes of data across multiple machines. Given the limited network capacity in data centers, it is essential to find fair yet efficient mechanisms for allocating network bandwidth to tasks with different usage characteristics and performance requirements. Furthermore, common communication patterns, such as multicasting, shuffling, and incasting must be considered in solving the problem. Many research challenges need to be addressed in order to effectively support these communication patterns so as to achieve fairness while minimizing the total network cost.

4.4 Performance-aware resource allocation

Another challenge concerns job performance requirements. As described in Sect. 3.4, jobs in production MapReduce environments often have different performance objectives in terms of job completion time. In this context, it is essential to find suitable resource allocations to ensure the performance objectives of each job are achieved. However, little work so far focused on developing performance models for MapReduce jobs, which makes performance management a difficult task. There are two important applications of a MapReduce performance model: (1) estimating the completion time and cost of a given job; and (2) finding appropriate resource allocation to satisfy job completion time constraints while minimizing total resource cost. Both applications are of crucial importance to job owners. However, developing an accurate performance model for MapReduce jobs is not easy, as it requires a careful modeling of every aspect of

a MapReduce job, such as resource requirements, machine capacity and capability, location of input data, failure rates, and dynamic network conditions. Devising such a comprehensive performance model is a clear research opportunity.

5 Representative literature review

The goal of this section is to survey some of the prominent works related to MapReduce, and how they address the challenges outlined in Sect. 4.

5.1 Job scheduling

Zaharia et al. [27] designed a new scheduling algorithm called Longest Approximate Time to End (LATE), that is robust to heterogeneity in cluster machines. The LATE scheduler uses a simple heuristic to estimate the progress of each task and launches speculative copies of tasks that take longest time to finish compared to other tasks of this job on fast machines. LATE has been incorporated into the current version of Hadoop. In [28], a policy called delay scheduling was designed to improve data locality. The job that should be scheduled next does not launch tasks on a node that does not have its file blocks. It waits for a small amount of time to find opportunities to schedule on local machines. To avoid starvation, the job is allowed to start nonlocal tasks if it has been waiting for a long time. This policy effectively improves data locality and job performance.

Quincy [18] is developed based on Microsoft Dryad framework. It encodes the scheduling decisions as a flow network where edge weights represent the demands of job scheduling. In this way, the cost of each scheduling decision is quantified in terms of the cost for data transfer and the overhead of terminating tasks. Each time a scheduling decision needs to be made (for example, a new job arrives), a min-cost flow algorithm [24] is used to find a minimum cost assignment. Quincy then kills some of the running tasks and launches new tasks to place the cluster in the configuration returned by the algorithm. Each Dryad job has a root task which monitors execution of other tasks, so the job scheduler knows fine-grained information about the tasks. Before making a decision, Quincy compares different types of costs, such as the cost of transferring 100MB data across certain switching layers and the cost of killing a task that has been running for 60 seconds. In order for the comparison between different types of costs to be reasonable, several parameters have to be carefully adjusted according to the settings of the cluster as well as the characteristics of the workloads. This can be a limitation of Quincy in some cases. Quincy, as well as delay scheduling try to improve job schedulability and resource utilization by designing or improving allocation policies, that is, how to assign tasks to resources.

In addition to allocation policies, post-scheduling actions can further help improve performance. Ananthanarayanan et al. [2] pointed out that heterogeneous machines and data skew can cause stragglers that significantly prolong job completion in an operational MapReduce cluster at Microsoft. They presented Mantri, a system that monitors tasks and reduces outliers using cause- and resource-aware techniques. Mantri's strategies include restarting outliers, network-aware placement of tasks and protecting outputs of valuable tasks. Similar to LATE, Mantri preempts and restarts a task elsewhere if doing so can improve the task running time. Compared to LATE, Mantri further considered the impact of network congestion on tasks progress, which makes it achieve better performance since currently most jobs are data intensive in MapReduce clusters. In the experiments reported in [7], we observed that the average network bandwidth between nodes could be as low as 40–50KB per second on Amazon EC2 [1], although the data transfer speed can be up to 10MB per second when the cluster is not busy and many nodes in the same rack are competing for bandwidth.

Lastly, as MapReduce clusters are shared by jobs with different priorities there is a need for designing preemption policies that achieve high performance while minimizing the negative impact of preemptions. Our previous work on MapReduce scheduling deals with this issue [7]. During our research experiments and in particular our experience with Google's production clusters, from time to time we have observed that some tasks with long running time are killed repeatedly when large production jobs arrive. Specifically, the arrival of a large production job can significantly cut down the share of each nonproduction job, resulting in large number of tasks being preempted. Since production jobs usually have short response time, after their completion the low-priority jobs (e.g., nonproduction jobs) are allowed to launch more tasks again. But before a long task can finish, it is often killed again when the next large production job arrives. Consequently, jobs consisting of these long tasks are heavily delayed and the resources allocated for their execution are wasted. We proposed a simple mechanism that works in conjunction with existing job schedulers to address this problem and implemented it on top of Apache Hadoop. Our technique, called Global Preemption (GP), consists in judiciously selecting tasks to be preempted by job schedulers in order to reduce the cost of preemption. The gist of GP is to trade short-term fairness for better efficiency. In the existing implementations of Hadoop job schedulers, when a large production job arrives, the scheduler will first compute the share of each job, and then kill tasks beyond each job's fair share to release slots when there is a shortage of slots to accommodate this job. Usually, a certain number of most recently launched tasks of each job will be killed. With GP, instead of killing newly launched tasks of each individual job,

the policy is to globally select the most recently launched from all the running tasks rather than from an individual job to minimize the cost of preemption. GP was evaluated under various types of workloads in Amazon EC2 and was able to improve system normalized performance by 15% during busy periods by effectively avoiding unnecessary preemptions while preserving fairness.

5.2 Data and task placement

Optimizing data and task placement is an important concern in MapReduce clusters. For example, Mantri [2] studied the problem of network-aware task placement using a simple greedy algorithm. More recently, Balaji et al. proposed Purlieus [23], a resource allocation system that uses a set of heuristics for optimizing the placement of both data and tasks. Purlieus first categorizes jobs into *map-input heavy* jobs and *reduce-input heavy* jobs, based on the input size of their map and reduce tasks. For each job category, Purlieus decides the placement of input data and then determines the placement of tasks. For map-input heavy jobs, Purlieus simply places data based on storage utilization and expected load on each machine. Afterward, a greedy algorithm is used for placing tasks in order to minimize the distance between task and input data. For reduce-input heavy jobs, Purlieus first places data close to each other to account for shuffling traffic generated for reduce tasks. An approximation algorithm is used to find a densely connected subgraph for data placement. Once data placement is determined, task placement can be performed greedily similar to the case of map-input heavy jobs. Experiments show Purlieus is able to achieve significant reduction (70% in some cases) in cross-rack traffic.

A related problem is data replication, which can be described as a dynamic data placement problem. Data replication is an important technique for improving job performance, because when multiple jobs need to access the same input data, the disk I/O of the machine possessing the data can become a performance bottleneck. By effectively replicating the data, this performance bottleneck can be alleviated. Recently, Ananthanarayanan et al. proposed Scarlett [3] as a framework for dynamic data replication in MapReduce clusters. The authors first observed uneven distribution of file popularity in production MapReduce clusters, and found that file popularity may change over time. Based on these observations, Scarlett dynamically replicates each data block using a prediction for file popularity. The placement of replicas is decided based on the expected load of each machine. To reduce the overhead of dynamic data replication, Scarlett employs (1) a multisource replication scheme similar to peer-assisted content distribution in P2P networks, and (2) data compression techniques. Experiments show that Scarlett significantly mitigates disk hotspots and improves job completion time by 20%.

5.3 Resource sharing

As mentioned in Sect. 3.4, heterogeneous jobs and machine capacities intensify resource contention, a major cause of performance degradation in MapReduce environments. Existing work so far primarily focused on one resource type, such as memory, among multiple resource types in a single machine. Ghodsi et al. [13] recently introduced a new fairness criterion called Dominate Resource Fairness (DRF). DRF is essentially a generalization of max-min fairness to multiple resource types. As different tasks have different resource bottlenecks (called dominate resource), DRF essentially ensures the share of the dominate resource is allocated according to the max-min principle. The authors further showed that compared to many other fairness criteria, DRF is Pareto-efficient, strategy-proof, envy-free, and incentivize users to share their resources. Experiments show that DRF can significantly outperform the current slot-based allocation scheme in terms of throughput and job completion time. Another potential performance bottleneck is the underlying data center network. It has been shown that data transfer across the network is a contributor to job completion time. As mentioned previously, much work has been carried to improve data locality and to avoid heavily loading the network (e.g., [28]).

More recently, Chowdhury et al. [8] studied the problem of optimizing data transfers for MapReduce clusters. The authors first pointed out that network scheduling should be carried out at data transfer level rather than individual flow level. Generally speaking, data transfer for MapReduce jobs can be divided into 3 types: broadcast, shuffle, and incast. The authors presented several techniques for improving the performance of these 3 data transfer types. Specifically, peer-to-peer based content distribution techniques have been used to improve communication locality. A weighted transfer scheduling mechanism was also devised to prioritize data transfers in order to reduce job competition time. However, much work still needs to be done for optimizing network transfers especially while taking into account the performance requirements of individual jobs.

5.4 Performance-aware resource allocation

As discussed in Sect. 4.4, developing performance models for predicting job performance and managing resource allocation is a key research challenge in MapReduce clusters. Tian et al. [25] proposed a technique for estimating the completion time of MapReduce jobs using a simple performance model. Based on a few assumptions, their model breaks a map task into 4 phases: Read, Map, Partition/sort, and Combine. The cost of each phase is modeled as a function of the total number of map tasks in the job and number of slots available for the job. Similarly, a reduce task is divided

into 4 phases: Copy, Sort, Reduce, and Write Back. The actual values of parameters used in the cost model can be estimated using small scale experiments. Using this performance model, the authors formulated optimization problems for (1) optimizing job performance subject to budget constraint, and (2) minimizing the budget used subject to completion time constraint. Initial experiments show the prediction model achieves good accuracy.

Another related work [23] proposes to control the resource allocation (e.g., minimum number of map and reduce slots) of each job subject to service level objectives (SLOs). The authors developed an analytic model for task scheduling and derived upper and lower bounds on completion time for all three phases: map, shuffle, and reduce. The parameters of the model are determined either through historical logs or using small scale experiments. The scaling factors can be determined using linear regression analysis. Using the proposed model, the authors gave a polynomial time algorithm for determining the number of map and reduce slots required for guaranteeing the completion of a job before its deadline. Realistic experiments show the proposed model achieves high prediction accuracy. However, similar to the work in [25], existing MapReduce performance models typically do not consider the dynamics in MapReduce clusters, such as task failures and network conditions. Thus, it is still an open challenge to find more accurate and realistic job performance models for production environments.

6 Conclusions

Cloud computational models such as MapReduce are crucial to the operations of Cloud computing environment in their ability to effectively deal with enormous volumes of data and computations at large-scale. However, to design an effective MapReduce computation model, one must be aware of and capable of dealing with the heterogeneity of workloads and cluster machines in Cloud environments.

In this article, we have analyzed the various types of heterogeneity in Cloud computing systems. In terms of workloads, the challenges arise from the bimodal distribution of job lengths and sizes, the burstiness of job arrival rates, and the heterogeneous resource requirements of jobs. In terms of machines, their diverse capacities in CPU, memory, I/O speed, and network bandwidth have immediate implications on the management MapReduce clusters. Consequently, many research challenges are present in different aspects of Cloud operations including job scheduling, data and task placement, resource sharing, and performance-aware resource allocation. In job scheduling, heterogeneous job characteristics and requirements affect efficiency and fairness of job completions; in data and task placement, heterogeneity hinders job completion rate and increases communication overhead; in resource sharing, heterogeneous job requirements calls for flexible and job-specific capacity allocation to achieve optimal machine utilization; in performance-aware resource allocation, performance models need to be developed in order to ensure the performance objectives of each job are achieved.

In surveying representative state-of-the-art work, it is clear that although advances are made to partially address some of the outlined challenges, there is even more open challenges yet to be explored. As MapReduce is still a relatively new technology, we can expect increased interest and attention on its design in the coming years. We believe this is a highly active and relevant research topic with much room for scientific exploration, and its undertaking will undoubtedly lead to many exciting discoveries.

Acknowledgement This work was supported in part by the Natural Science and Engineering Council of Canada (NSERC) under its Strategic program, and in part by the WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education, Science, and Technology (Project No. R31-2008-000-10100-0).

References

1. Amazon EC2, http://aws.amazon.com/ec2/
2. Ananthanarayanan G, Kandula S, Greenberg A, Stoica I, Lu Y, Saha B, Harris E (2010) Reining in the outliers in MapReduce clusters using Mantri. In: Proc. OSDI
3. Ananthanarayanan G, Agarwal S, Kandula S, Greenberg A, Stoica I, Harlan D, Harris E (2011) Scarlett: coping with skewed content popularity in mapreduce clusters. ACM European conference on computing systems (EuroSys)
4. Apache hadoop, http://hadoop.apache.org/
5. Chen Y, Ganapathi AS, Griffith R, Katz RH (2010) Towards understanding cloud performance tradeoffs using statistical workload analysis and replay. Tech rep, University of California, Berkeley
6. Chen Y, Ganapathi AS, Griffith R, Katz RH (2010) Analysis and lessons from a publicly available Google cluster trace. Tech rep, University of California, Berkeley
7. Cheng L, Zhang Q, Boutaba R (2011) Mitigating the negative impact of preemption on heterogeneous MapReduce workloads. In: International conference on network and service management (CNSM)
8. Chowdhury M, Zaharia M, Ma J, Jordan M, Stoica I (2011) Managing data transfers in computer clusters with orchestra. In: ACM SIGCOMM
9. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
10. Feng H, Misra V, Rubenstein D (2005) Optimal state-free, size-aware dispatching for heterogeneous M/G-type systems. In: Performance evaluation
11. Foss S, Korshunov D (2006) Heavy tails in multi-server queue. In: Queueing systems: theory and applications
12. Ghemawat S, Gobioff H, Leung ST (2003) The Google file system. ACM SIGOPS Oper Syst Rev 37(5):29–43
13. Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I (2011) Dominant resource fairness: fair allocation of multiple resource types. In: Networked systems design implementation (NSDI), pp 323–336
14. Hadoop distributed file system, http://hadoop.apache.org/hdfs/

15. Harchol-Balter M (2002) Task assignment with unknown duration. J. ACM 49(2):260–288
16. Harchol-Balter M, Downey AB (1997) Exploiting process lifetime distributions for dynamic load balancing. In: ACM transactions on computer systems
17. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: Proc. Eurosys, March 2007, pp 59–72
18. Isard M, Prabhakaran V, Currey J, Wieder U, Talwar K, Goldberg A (2009) Quincy: fair scheduling for distributed computing clusters. In: Proc. SOSP
19. Jackson DS, Kunzinger FF (2003) Calculation of system availability using traffic statistics. Bell Labs Tech J 7(3):139–150
20. Lempiäinen J, Manninen M (2002) Radio interface system planning for GSM/GPRS/UMTS. Springer, Berlin
21. Mishra AK, Hellerstein JL, Cirne W, Das CR (2010) Towards characterizing Cloud backend workloads: insights from Google compute clusters. ACM SIGMETRICS Perform Eval Rev 37(4):34–41
22. NIST definition of cloud computing v15, http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc
23. Palanisamy B, Singh A, Liu L, Jain BP (2011) Locality-aware resource allocation for MapReduce in a cloud. In: ACM international conference on supercomputing (SC)
24. Tari Z, Broberg J, Zomaya A, Baldoni R (2005) A least flow-time first load sharing approach for distributed server farm. J Parallel and Distributed Computing
25. Tian F, Chen K (2011) Towards optimal resource provisioning for running MapReduce programs in public clouds. In: IEEE international conference on cloud computing (CLOUD)
26. Traffic analysis for voice over IP, Cisco Technical report, 2007
27. Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I (2008) Improving MapReduce performance in heterogeneous environments. In: Proc. OSDI
28. Zaharia M, Borthakur D, Sarma JS, Elmeleegy K, Shenker S, Stoica I (2010) Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proc. Eurosys
29. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 1(1):7–18