Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud

Gunho Lee[†], Byung-Gon Chun[‡], Randy H. Katz[†]

[†]*University of California, Berkeley,* [‡]*Yahoo! Research*

Abstract

Data analytics are key applications running in the cloud computing environment. To improve performance and cost-effectiveness of a data analytics cluster in the cloud, the data analytics system should account for heterogeneity of the environment and workloads. In addition, it also needs to provide fairness among jobs when multiple jobs share the cluster. In this paper, we rethink resource allocation and job scheduling on a data analytics system in the cloud to embrace the heterogeneity of the underlying platforms and workloads. To that end, we suggest an architecture to allocate resources to a data analytics cluster in the cloud, and propose a metric of share in a heterogeneous cluster to realize a scheduling scheme that achieves high performance and fairness.

1 Introduction

As the number and the scale of Internet services increases, more services are run in the cloud computing environment. Many of these services require large data processing to serve customers. They also generate a huge amount of data to be analyzed, to monitor and improve the quality. To minimize the response time of data processing queries, it is desirable to store and process data in the same cluster to exploit data locality.

Since capital and operational costs are primary considerations, we want to minimze the size of the cluster while meeting the performance requirements or the deadlines of queries. The resource demands for data analytics job fluctuate over time. The cloud computing environment provides dynamic provisioning [5]. Thus we can allocate just enough machines to store the data and add or remove machines according to the workload demands. We can either use the remaining machines for other purposes or not pay for them at all.

Data analytics workloads have heterogeneous resource demands because some workloads may be CPU-intensive whereas others are I/O-intensive. Some of them might be able to use special hardware like GPUs to achieve dramatic performance gains [9].

It is also likely that the computing environment is heterogeneous. The cloud consist of generations of servers with different capacities and performance; therefore, various configurations of machines will be available. For example, some machines are more suitable to store large data whereas others run faster computations. As the performance of a job depends on where it runs, we need to track *job affinity* or determine which type of machine offers the most suitable cost-performance trade-off for a job.

Accounting for heterogeneity properties in the cloud becomes more difficult when the cluster is shared among multiple jobs because the most suitable type of machine depends on the job. The data analytics system must provide fairness among jobs. For example, when multiple jobs are running and accessing the same set of data, some machines will offer the best performance for a job while not for the others. Other machines may show poor performance for all the jobs, but they may still be available (e.g., storage-oriented machines to store data). A key question is how to schedule jobs on these machines so that each receives its "fair" share of resources to make progress while providing good performance.

In this paper, we consider resource allocation and job scheduling problem of the data analytics cluster in the cloud. Given the fluctuating resource demands of data analytics workloads, we must scale the cluster according to demands. To that end, we suggest a resource allocation strategy that (1) divides machines into two pools - *core* nodes and *accelerator* nodes - and (2) dynamically adjusts the size of each pool to reduce cost or improve utilization.

In addition, to provide good performance while guaranteeing fairness in shared heterogeneous cluster, we propose *progress share* as a new fairness metric to redefine the share of a job in heterogeneous environment. We also show how to adapt it to a data analytics system.

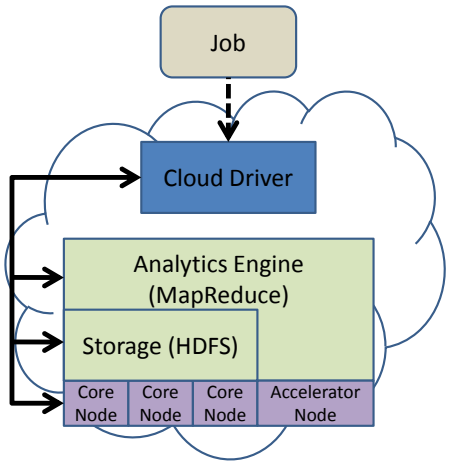

Figure 1: Data Analytic Cloud Architecture

To present use case, we use Hadoop as the data analytics system and Amazon EC2 as the cloud environment. However, the idea of exploiting heterogeneity to improve efficiency and fairness may be applied to other systems running in other environments.

Hadoop [3] is a widely-used data analytics system, which consists of Hadoop Distributed File System (HDFS) as a storage layer and and Hadoop MapReduce as an analytics engine. A file in HDFS is divided into blocks, and replicas of each block are stored on nodes in the cluster. On top of HDFS, Hadoop MapReduce is used to process data. Users submit jobs that consist of a map function and a reduce function to Hadoop MapReduce, then map and reduce tasks are launched on slots hosted on participating nodes process data stored in HDFS and aggregate results.

Amazon EC2 is a public cloud service that enables users to lease virtual machines. Amazon charges per hour and per machine for this public cloud service without requiring any long-term commitments. Users can choose the spec of their virtual machines from 11 instance types tagged with different prices. Some of instance types available are shown in Table 1.

This paper is organized as follows. We first describe a resource allocation strategy for a data analytics cluster in the cloud in Section 2. In Section 3, we present a scheduling scheme in heterogeneous and shared environments. We illustrate the benefits of our approach with case studies in Section 4 and conclude with future work in Section 5.

2 Resource Allocation

In this section, we will examine the resource allocation strategy to make data analytics in the cloud efficient. By "efficient", we mean minizing the size of the cluster while meeting the performance requirements. In this way, we can improve overall data center utilization by letting the remaining machines utilized by other applications, or reduce cost by not paying for redundant machines. To that end, we propose an architecture to build a data analytics system in the cloud as seen in Figure 1.

In this architecture, participating nodes are grouped into one of two pools: (1) long-living *core* nodes to host both data and computations, and (2) *accelerator* nodes that are added to the cluster temporarily when additional computing power is needed. An analytic engine (e.g., Hadoop MapReduce) runs on nodes in both pools whereas a storage system (e.g., HDFS) is deployed only on core nodes. This approach is similar to the previous work of Chohan et al. [6], in which spot instances (cheaper but may be terminated without notice) of Amazon EC2 are added to processing nodes to speed up Hadoop jobs. The *cloud driver* manages nodes allocated to the analytic cloud and decides when to add/remove what type of nodes to/from which pool, and how many.

Users submit a job to the cloud driver with a few hints about the job characteristics, including memory requirement, ability to use special features like GPUs, and the deadline, if available. Many production queries are routinely processed, so the cloud driver keeps the history of query executions to estimate the submission rate of these queries and update the hints provided. It also monitors the storage system to estimate the incoming data rate. In this way, the cloud driver predict the resource requirements to process queries and to store data.

The cloud driver is responsible for allocating resources to the cluster. The number of core nodes is determined primarily based on the required storage size. In addition, the cloud driver refers to the history to see if more nodes should be added to the core pool to accommodate the production queries. When many production queries with tight deadlines are anticipated or a large ad-hoc query is submitted, the cloud driver will add nodes to the accelerator pool temporarily to handle them rather than allocating too many core nodes that will be underutilized.

When adding nodes, the cloud driver also makes a decision on which resource container (e.g., virtual machine) to use. As an illustration, we examine the case when we use Amazon EC2 [1] for the cloud.

If we consider only the cost for the storage, using m1.large instances is the cheapest. However, these instances also have the least computing power among instance types available , so we might need more nodes to accommodate the production queries. In this case, using other instance types such as c1.medium can be more efficient in terms of the whole expense. Moreover, some jobs may run significantly faster on nodes of a particular instance type (e.g., cg1.4xlarge instances with GPUs). Hence, it is important to know the job/instance type relationship (which we call *job affinity*) to find a good mix

Instance Type	$/Hour	Disk(GB)	$/GB/mon	Core	CU	$/CU	Mem(GB)	GB/Core	I/O
m1.small	0.085	160	0.38	1	1	61.20	1.7	1.70	moderate
m1.large	0.340	850	0.29	2	4	61.20	7.5	3.75	high
m2.2xlarge	1.000	850	0.80	4	13	55.38	34.2	8.55	high
c1.medium	0.170	350	0.35	2	5	24.48	1.7	0.85	moderate
cc1.4xlarge	1.600	1690	0.68	8	33.5	34.39	23	2.88	very high
cg1.4xlarge	2.100	1690	0.89	8	33.5	45.13	23	2.88	very high

Table 1: Example of EC2 Instances. CU represents Compute Unit.

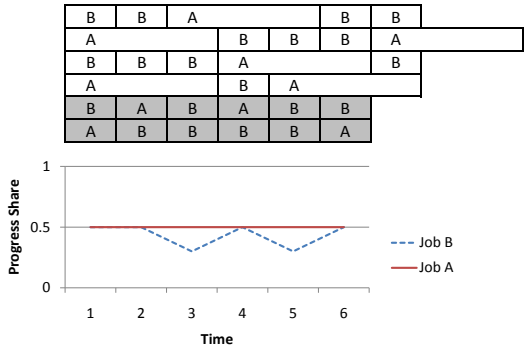

Figure 2: Scheduling based on slot share

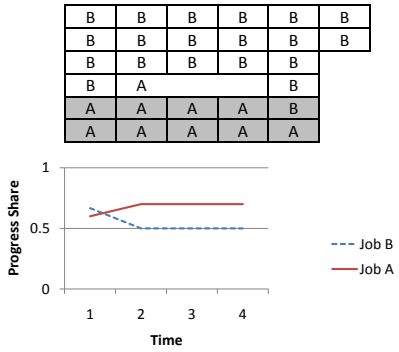

Figure 3: Scheduling based on progress share

of different instances that minimize the cost to maintain the cluster.

We quantize *job affinity* using the relative speed of each instance type for a particular job, which we call *computing rate*(CR). $CR(i, j)$ is the computing rate of instance type i for job j. If $CR(i_1, j)$ is twice of $CR(i_2, j)$, a task of job j runs twice faster on i_1 than on i_2, or finishes in a half of time. By using the computing rate information and the cost of each instance type, the cloud driver can make a decision on which instance type to use. CR is determined during the calibration phase of a job execution, which is described in Section 4.

3 Scheduling

Once the cloud driver allocates a set of resource units such as virtual machines, the analytics engine uses the resources that are heterogeneous and shared among multiple jobs. In this section, we consider issues in job scheduling on a shared, heterogeneous cluster, to provide good performance while guarenteeing fairness.

3.1 Share and Fairness

In a shared cluster, providing fairness is one of the important features that the analytics engine should support. There are many ways to define fairness, but one method might be having each job receive equal (or weighted) share of computing resources at any given moment. In that sense, Hadoop Fair Scheduler [4] takes the number of slots assigned to a job as a metric of share, and it provides fairness by having each job assigned the same number of slots.

In a heterogeneous cluster, the number of slots might not be an appropriate metric of the share because all the slots are not the same. Moreover, even on the same slot, the computation speed varies depending on jobs. The performance variance on different resources is also important to consider to improve overall performance of the data analytics cluster; assigning a job to unpreferred slots will not only make the job run slow, but also may prevent other jobs that prefer the slot from utilizing it.

To realize fair and effective job scheduling in shared and heterogeneous cluster, we introduce *Progress Share*(PS) that captures the contribution of each resource to the progress of a job. We also show how to adopt it using the Hadoop MapReduce as an example analytics engine.

3.1.1 Progress Share

Conceptually, progress share refers to how much progress each job is making with assigned resources (or slots in Hadoop) compared to the case of running the job on the entire cluster without sharing; therefore, it is between 0 (no progress at all) and 1 (all available resources are occupied). The computing rate (CR) is used to calculate the progress share of a job.

Specifically, given that $CR(s, j)$ is the computing rate of slot s for job j, the progress share $PS(j)$ of job j is defined as follows:

$$PS(j) = \frac{\sum_{s' \in S_j} CR(s', j)}{\sum_{s'' \in S} CR(s'', j)} \quad (1)$$

where S_j is a set of slots running tasks of job j and S is the set of all slots. The sum of the progress share for all jobs indicates the effectiveness of the resource assignment. If it is below 1, there is an alternative assignment that makes the sum above or equal to 1.

For example, suppose that two jobs, A and B, run on a cluster that consists of two different types of nodes, $N1$ and $N2$, where there are two slots of $N1$ and four slots of $N2$. In addition, assume that a task of job A runs three times faster on a slot of $N1$ than $N2$, whereas job B runs on $N1$ as fast as $N2$. Figure 2 is an example of scheduling when the number of assigned slots is used as a share metric. Gray and white cells represent the slots of $N1$ and $N2$, respectively. The letter in each cell indicates the job occupying the slot. The graph below shows how the progress share of each job changes over time. (It is drawn only to the point at which there are enough tasks to schedule.) Even though each job occupies the same number (three) of slots at all times, the progress share of job A often falls below its fair share (0.5) because many tasks of job A run on slots of $N2$, which is not suitable for the job. As seen in Figure 3, a progress share-based scheduling guarantees that each job receives its fair share. It also reduces the job finishing time, thereby improving overall performance.

3.1.2 Scheduler

To calculate the $ProgressShare$ of each job, the analytics engine scheduler should be aware of the per-slot computing rate(CR). To that end, each job goes through two phases: calibration and normal. When a job is submitted, it starts with the calibration phase. In this phase, at least one map task is launched on each type of node. By measuring the completion time of these tasks, the scheduler can determine the CR. Once the scheduler knows the CR, the job enters the normal phase.

During the normal phase, the scheduler works similar to the Hadoop fair scheduler. When a slot becomes available, a job of which the share is less than its minimum or fair share is selected. However, if there is another job with a significantly higher computing rate on the slot, the scheduler chooses that job to improve overall performance. This is similar to the "delay scheduling" [10] mechanism in Hadoop fair scheduler.

By using progress share, the scheduler can make an appropriate decision. As a result, the cluster is better utilized (i.e., the sum of the progress share ≥ 1). In addition, each job receives a fair amount of computing resources.

4 Case Study

In this section, we examine two case studies to illustrate the potential benefits of our system in Amazon EC2.

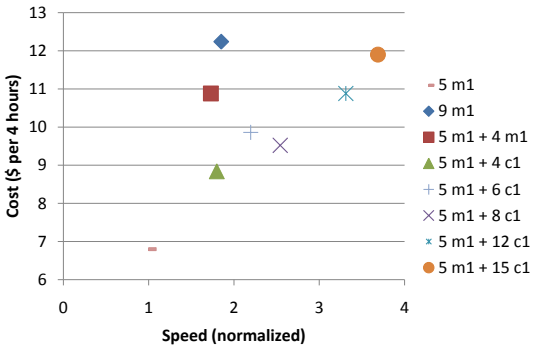

Figure 4: Cost and speed of various configuration

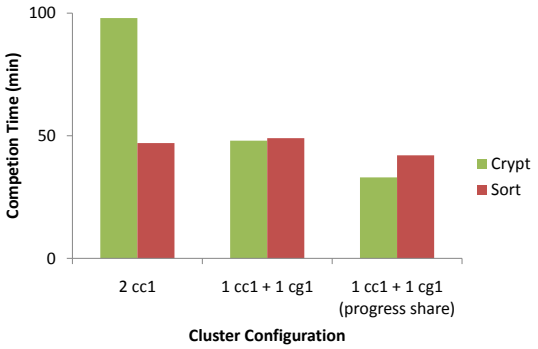

Figure 5: Accelerable jobs and the progress share

Resource Allocation and Accelerator Nodes First, we consider a case in which a number of production queries are issued every 4 hours. We chose 5 jobs from the gridmix2 benchmark [2] as production queries and ran them on the clusters with various configurations.

Figure 4 shows the cost to maintain the cluster during one round of the production queries (4 hours) and the relative speed of processing the queries. In this figure 4, "9m1" means using 9 m1.large instances as core nodes, "5m1+4c1" represents a cluster with 5 m1.large core nodes and 4 c1.medium accelerator nodes, and so on. In this experiment, the capacity of the 5m1 configuration was overloaded by the queries; thus it took more than 4 hours. With three of the accelerator-rich configurations (5m1+8c1, 5m1+12c1, and 5m1+15c1), it took less than 2 hours, so accelerator nodes in these configurations were released after the queries were done. The other configurations completed the queries in between 2 and 3 hours.

Note that c1.medium instances provide higher performance per cost because they have faster CPUs at lower prices than do m1.large (see Table 1). However, they are equipped with less memory; therefore, they might be of no use for jobs that require a large amount of memory. The graph also points out that using more accelerators can cost less, while allowing jobs to be completed faster. Because the cluster with 6 accelerator nodes of c1.medium instance can finish all jobs less than 2 hours,

we need to pay for only 2 hours' usage of accelerator nodes. On the other hand, it takes more than 2 hours with 4 c1.medium accelerators, which requires paying charges for 3 hours. In this case study, we observed that there are a number of ways to allocate resources with cost-performance trade-offs. Making the right decision maintains the cluster cost effectively.

Job Affinity and Progress Share The second case is when there is significant job affinity. We used a GPU-accelarable dictionary-based cryptographic attack job (Crypto), and a sort job (Sort) for this case. We prepared a small cluster consisting of two nodes and used the fair scheduler.

The first bars in Figure 5 show the completion time of each job when the two nodes are all cc1.4xlarge instances. For the second bars, we replaced a node with a cg1.4xlarge instance that is identical to cc1.4xlarge except for the GPUs with which it is equipped. Here, we can observe a significant performance gain of using GPUs. This suggests the benefit of having a heterogeneous cluster. However, just having a heterogeneous cluster does not lead to an optimal usage; the scheduler will assign tasks randomly without being aware of job affinity. The last bars show the results with a hardware-aware scheduler that adopts progress share. By prioritizing to the Crypto job to the nodes with GPUs, the job was completed much earlier. It is worth noting that even the sort job finished earlier compared to other cases because the Sort receives more than half of slots at any moment. Assigning a fraction of a cg1.xlarge node to Crypto contributes a significant amount towards its progress share, which in turn makes the Sort job receive more resources to match its progress share to Crypto's. Even if Sort receives more slots than Crypto, we argue that it is more fair than giving the same number of slots to each job (as the current Hadoop does) because Crypto already receives significantly preferred resources, which quickens its progress. In this case study, we can see that the scheduling algorithm of the analytics engine plays an important role to improve performance while providing fairness.

5 Conclusion

The cloud environment provides heterogeneous hardware and resource demands; therefore, it is important to exploit these features to make a data analytics cluster in the cloud efficient. In this paper, we presented a system architecture to allocate resources to such a cluster in a cost-effective manner, and discussed a scheduling scheme that provides good performance and fairness simultaneously in heterogeneous cluster, by adopting progress share as a share metric.

Beyond data analytics systems, many systems running in the cloud involve multilevel scheduling - resource allocation at infrastructure level and job scheduling at application level. Mesos [8] takes this approach to provide resource sharing and isolation across distributed applications. It adopts *Dominent Resource Fairness*(DRF) [7] as an example of resource allocation policy and leaves application level scheduling to each application. We plan to experiment with PS in a framework to be constructed on top of Mesos.

References

[1] Amazon ec2. `http://aws.amazon.com/ec2`.

[2] Gridmix. `http://hadoop.apache.org/mapreduce/docs/current/gridmix.html`.

[3] Hadoop. `http://hadoop.apache.org`.

[4] Hadoop fair scheduler. `http://hadoop.apache.org/common/docs/r0.20.1/fair_scheduler.html`.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[6] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See spot run: Using spot instances for mapreduce workflows. In *HotCloud*, 2010.

[7] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

[8] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[9] J. Polo, D. Carrera, Y. Becerra, V. Beltran, and J. T. andEduard Ayguad. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In *39th International Conference on Parallel Processing (ICPP2010)*, 2010.

[10] M. Zaharia et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys*, 2010.