Scaling MapReduce Applications across Hybrid Clouds to Meet Soft Deadlines

Michael Mattess, Rodrigo N. Calheiros, and Rajkumar Buyya
Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: mmattess@csse.unimelb.edu.au, {rnc, rbuyya}@unimelb.edu.au

Abstract—**Cloud platforms make available a virtually infinite amount of computing resources, which are managed by third parties and are accessed by users on demand in a pay-per-use manner, with Quality of Service guarantees. This enables computing infrastructures to be scaled up and down accordingly to the amount of data to be processed. MapReduce is among the most popular models for development of Cloud applications. As the utilization of such programming model spreads across multiple application domains, the need for timely execution of these applications arises. While existing approaches focus in meeting deadlines via admission control or preemption of lower priority applications, we propose a policy for dynamic provisioning of Cloud resources to speed up execution of deadline-constrained MapReduce applications, by enabling concurrent execution of tasks, in order to meet a deadline for completion of the Map phase of the application. We describe the proposed algorithm and an actual implementation of it in the Aneka Cloud Platform. Experiments on such prototype implementation show that our proposed approach can effectively meet the soft deadlines while minimizing the budget for application execution.**

I. Introduction

In recent years, we have been observing an increased importance in the capacity of collecting, storing, and processing large amounts of data. This trend is noticed both in academia and industry. Technological advances such as high-capacity and high-speed storage devices make quicker and cheaper the storage and the access of such information. Information collected from a range of sources has to be combined and analyzed so relationships between different elements can be inferred from the data. As the typical goal of such processing is the discovery of relationships that can give competitive advantage to companies, or scientific breakthroughs to researchers, there is an increasing importance in the capacity of timely analysis of large amounts of data.

Until recently, the preferred method for increasing processing capacity was the deployment of large scale Clusters. However, expansion of the Cluster capacity demands huge investment in procurement, maintenance, and management of the infrastructure, besides extra costs related to powering and cooling such infrastructures.

An alternative to Clusters is Cloud computing [1]. Cloud platforms make available a virtually infinite amount of computing resources, which are managed by third parties and are accessed by users on demand in a pay-per-use manner, with Quality of Service guarantees given by Cloud service providers. This enables the computing infrastructure to be scaled up and down accordingly to the amount of data to be processed. Nevertheless, even though the hardware infrastructure is available that supports such requirements, applications performing data analytics still need better tuning in order to fully utilize the processing power supplied by Cloud infrastructures.

Among various available programming models, MapReduce [2] is one of the most suitable for development of applications deployed on Clouds. MapReduce is a parallel computing model designed for processing of large amounts of data in large computational infrastructures. A MapReduce application consists of two types of tasks, namely *Map* and *Reduce*. When a MapReduce application starts, Map tasks process input data and generate intermediate data, which is structured as key-value pairs. Reduce tasks then combine all the values associated with a key and emit the application's output data.

As the utilization of the MapReduce model spreads across multiple application domains, the need for timely execution of MapReduce applications arises. Existing approaches for execution of such *deadline-constrained MapReduce applications* focus in meeting deadlines via admission control or preemption of lower priority applications. However, they cannot solve conflicts when applications with the same priority have to be concurrently executed.

To tackle this limitation of current approaches for the problem, we propose a novel policy for *dynamic provisioning of public Cloud resources to speed up execution of MapReduce applications*. We target the Map phase as the target of the soft deadline because this phase contains tasks that generally have uniform execution time.

In this paper, we describe the proposed algorithm and detail an actual implementation of it in the Aneka Cloud Platform [3]. Experiments show that our proposed approach can effectively meet the soft deadlines while minimizing the budget for application execution.

II. Related Work

The MapReduce programming model [2] and implementations have been extensively used by companies and academia. An Open Source implementation of the programming model is Hadoop[1], which has been used as a baseline for improvements

[1]http://hadoop.apache.org/

in the implementations. Most existing MapReduce research target large scale, single site environments [4]–[6], whereas we target a hybrid Cloud infrastructure where in-house resources are used together with public Cloud resources to meet application soft deadlines.

Matsunaga *et al.* [7], Polo *et al.* [8], Luo *et al.* [9], and Fadika *et al.* [10] proposed models for execution of MapReduce applications across multiple Clusters. However, resources from Clusters are accessed on a best-effort basis, with the intention of speeding up application execution. This makes the proposed infrastructure similar to Grid computing infrastructures. Our approach, on the other hand, scales the computation across a public Cloud, where resources can be virtually scaled unlimitedly, with the goal of meeting a user-defined soft deadline.

About execution of MapReduce applications in Clouds, Tsai *et al.* [11] proposed a model for replication of executors for MapReduce tasks. However, it does not present specific strategies for provisioning resources for computational tasks, while our proposal manages both provisioning of resources and actual scheduling of tasks for MapReduce computation.

Tian and Chen [12] and Verma *et al.* [13] independently proposed models for optimal resource provisioning for running MapReduce applications in public Clouds, whereas Rizvandi *et al.* [14] proposed a method for automatic configuration of MapReduce configuration parameters in order to optimize execution of applications in a Cloud. The models, however, were not develop to address hybrid Clouds elements, such as the existence of private Cloud nodes available for computation, which is the target environment of our proposed approach.

Sehgal *et al.* [15] developed an interoperable implementation of MapReduce able to execute applications on Clusters, Grids, and Clouds. The main motivation of such a system is to enable interoperation of applications that are strongly tied to a given infrastructure, while we deploy hybrid infrastructures with the intention of meeting deadlines of applications.

Dong *et al.* [16] proposed a two-level scheduling approach for meeting deadlines of real-time MapReduce applications running concurrently to non-real-time applications. Their approach prioritizes real-time applications over non real-time ones, but does not dynamically provision extra resources for meeting application deadline as does our approach.

Kc and Anyanwu [17] proposed an approach were an admission control mechanism rejects requests for executing MapReduce applications when deadlines cannot be met. Instead of rejecting requests, our approach utilizes dynamic provisioning for allocating extra resources.

To the best of our knowledge, no current approach for execution of MapReduce applications is able to scale applications across hybrid Clouds in order to meet application soft deadlines as does the policy we propose in this paper.

III. Motivation and Application Scenario

MapReduce [2] is a programming model that consists of two types of tasks, namely *Map* and *Reduce*. When a MapReduce application starts, Map tasks process input data and generate intermediate data, which is structured as key-value pairs. Reduce tasks then combine all the values associated with a key and emit the application's output data. Reduce tasks cannot start to execute until all Map tasks have completed.

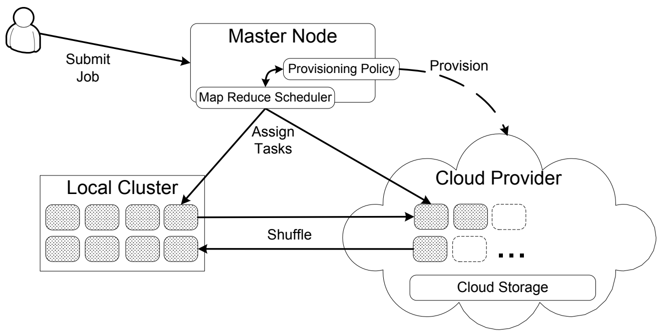


Fig. 1. The proposed application scenario for the deadline-aware execution of MapReduce applications on hybrid Clouds.

This class of applications became popular with the increasing amount of storage available in public and private Clouds, and has been widely applied for processing large amounts of data using an equally large amount of work units. While existing approaches focused in meeting deadlines of MapReduce applications via admission control or preemption of lower priority applications (as detailed in the previous section), we propose a novel approach for addressing this challenge by *dynamically provisioning public Cloud resources to speed up execution of MapReduce applications via a resource provisioning policy*.

Figure 1 depicts the envisioned scenario for the proposed policy. A user submits a MapReduce application to a local master node able to manage provisioning of resources and scheduling of MapReduce tasks. These tasks are executed by worker nodes that compose the local Cluster (private Cloud). The local Cluster is complemented with dynamically provisioned resources from a public Cloud to speed up execution of tasks. Provisioning of such public Cloud resources is subject to a Provisioning policy that is proposed in this paper.

In our envisioned scenario for deadline-constrained execution of MapReduce tasks, users are able to set a soft deadline for the completion of the Map phase. We apply this model of guaranteeing deadlines only for the Map phase because all Map tasks use the same code and operate on similar amounts of data; hence they require similar amounts of time to execute. This means that after a few Map tasks complete, it is possible to predict the total amount of time that will be needed for the remaining Map tasks. For the case of Reduce tasks, on the other hand, execution time of tasks cannot be predicted until the Map phase has completed, as assignment of key-values to Reduce tasks, and the amount of computation associated to such value, tend to exhibit higher variance in execution time. A deadline for the completion of the entire application, on the other hand, would require a technique for splitting the deadlines of each stage. A method for best performing this split is subject of future work and in this paper the focus is on dynamic provisioning of public Cloud resources to meet the Map phase deadline.

Once the deadline-constrained application is submitted for execution, Map tasks are deployed in the local available resources and in resources dynamically provisioned to speed up the task. The Map phase and Reduce phase are considered as two distinct phases and are investigated separately. During the Map phase, provisioning of additional resources is examined and the supplied deadline is used as a guideline for the completion of the Map phase. In the case of the Reduce phase, the examination focuses on how to schedule the tasks in order to utilize the additional resources that may have been requested during the Map phase.

We assume that virtual machine images supporting the execution of Map and Reduce tasks, as well as the dataset to be processed by the MapReduce application, are present on the storage platform of the public Cloud provider. This assumption is reasonable considering that public Clouds provide reliable storage services that can be used for guaranteeing data redundancy and backup. This extra reliability also justifies the incurring storage cost to keep such data in the public Cloud.

As the proposed dynamic provisioning techniques are applied on individual MapReduce applications, the sequence of events from application submission to the application execution is summarized in the following steps:

1) The initial state of the system consists of local worker nodes registered with the master node and ready to execute MapReduce tasks. Datasets are available for local workers and are also stored in the Cloud provider's storage service as backup replicas and also for being able to be used by dynamically provisioned workers;
2) A MapReduce application is submitted to the master node and the scheduler begins assigning Map tasks to worker nodes;
3) As Map tasks complete, the scheduler assigns new Map tasks to idle workers;
4) When a predefined number or fraction of the Map tasks completes, the master uses the provisioning policy and, based on the application deadline, requests additional resources from a Cloud provider as defined by the policy;
5) Requested resources register with the master node as they become available and the scheduler assigns Map tasks to them;
6) When the Map phase completes, the scheduler begins assigning Reduce tasks to workers;
7) Each Reduce task obtains the intermediate data to be reduced from other nodes;
8) The output of the Reduce tasks may remain on the nodes in anticipation of another MapReduce application, which will further process the data. Alternatively, the output can be collected by the master node and sent back to the user that submitted the application.

Finally, it is important noting that, because utilization of public Cloud resources incurs a financial cost to the requester, the proposed provisioning policy aims at allocating the *minimum number of resources able to meet the application deadline*. In the next section the proposed policy that enables such envisioned scenario is detailed.

IV. Deadline-Aware Dynamic Provisioning for MapReduce applications

The proposed provisioning policy aims at meeting MapReduce application deadlines at the minimum possible cost for utilization of public Cloud resources. Because the deadline is defined as the time for the Map phase of the application to complete, the policy requests just enough resources so that the Map phase will be completed before its deadline. Since all Map tasks execute the same code and operate on (almost) the same amount of data, it is reasonable to expect that the remaining Map tasks will require a similar amount of time. Hence, the provisioning policy makes use of the turnaround time of the first batch of Map tasks that were assigned to local resources to estimate the execution time of remaining tasks. The other parameter to the provisioning policy is the amount of time remaining until the deadline.

The provisioning policy can be further customized via tunable parameters:

- *MARGIN* is a 'safety margin' that is removed from the remaining time to account for errors in the prediction of tasks execution time.
- *LOCAL_FACTOR* is a multiplier applied to the average run time of the first batch of Map tasks to generate an estimation of the expected Map task execution time on the Cluster considering variation in the worker performance.
- *REMOTE_FACTOR* is similar to the LOCAL_FACTOR, but is set to predict the expected Map task execution time on public Cloud resources. This allows accounting for the expected variation in the performance of local and public Cloud resources: as the underlying infrastructure is not under control of our system, and because such resources are typically shared among public Cloud customers (multitenancy), it is expected that public Cloud resources experience a higher variation in performance than local resources.
- *BOOT_TIME* is the expected amount of time between requesting a new resource from the public Cloud and that resource becoming ready to execute tasks.

The decision of number of public Cloud resources to be allocated to a MapReduce application is detailed in Algorithm 1. It operates by deducting the MARGIN from the remaining time for execution of the application. The estimated execution time of Map tasks is then updated for both tasks running in the local Cluster and in remote Clouds, via application of correspondent factors. Next, the algorithm conservatively computes the number of tasks that can be executed locally before the deadline. The reverse calculation is then applied for the exceeding tasks to determine the number of Cloud resources to be provisioned for the rest of tasks. In this calculation, the boot time of Cloud resources is also considered.

For the Reduce phase, three basic approaches for using the available resources were examined. These policies determined

Algorithm 1: Provisioning Policy

```
Data: $secRemaining$: Available time until the deadline.
Data: $avgRunTime$: Measured execution time of Map tasks.
1  $secRemaining = secRemaining - MARGIN$;
2  $localRT = avgRunTime \times LOCAL\_FACTOR$;
3  $remoteRT = avgRunTime \times REMOTE\_FACTOR$;
4  $LocalRemainingTime = secRemaining - localRT$;
5  $LocalTasksPerBox = \lfloor \frac{LocalRemainingTime}{localRT} \rfloor$;
6  $intLocalTaskCount = LocalTasksPerBox \times resourceCount$;
7  $RemoteTasks = mapTasksRemaining - LocalTaskCount$;
8  $RemoteTotalRunTime = RemoteTasks \times remoteRT$;
9  $RemResCount = \frac{RemoteTotalRunTime}{secRemaining-(BOOT\_TIME)}$;
10 $extraRes = \lceil RemResCount \rceil$;
11 if $RemoteTasks > 0$ then
12     $RequestResources(extraRes)$;
13 end
```

which resources were used to execute the Reduce tasks:

- The *Local Only* policy forces all the Reduce tasks to be executed on local resources;
- The *Remote Only* policy forces all Reduce tasks to be executed on public Cloud resources; and
- The *Hybrid* policy makes use of all the available resources to execute Reduce tasks.

In Section VI we examine the impact of the different policies in the overall performance of the dynamic provision policy.

V. Implementation

The approach described in the previous section was implemented in the Aneka Cloud Platform [3], which has an experimental MapReduce implementation [18]. Aneka is a software platform and a framework for the development and deployment of distributed applications in public and private Clouds. Aneka is designed as a container in which a number of services can be configured to execute. For example, a worker node might have been configured to execute both the MapReduce service and a Bag-of-Tasks application execution service. The master node on the other hand might only run the corresponding scheduling services.

Because the initial MapReduce implementation of Aneka supported only execution of applications in a single domain, and without dynamic provisioning, it was extended to make use of the new provisioning service. In addition, a number of changes were applied to such MapReduce implementation to enable it to run across different domains. Next, we detail each major change.

Dynamic Provisioning

One of the built in services of the Aneka platform is the Cloud provisioning service [19]. This service provides a consistent interface to other parts of the Aneka system, while also permitting different Cloud providers to be used. It has two main components, the individual *Resource Pools* and the *Resource Pool Manager*. *Resource Pools* are responsible for interacting with Cloud providers and managing virtual machines from a given provider. Hence, there is one implementation of the *Resource Pool* for each supported Cloud provider. A number of *Resource Pools* can be used concurrently, and they are all controlled by the *Resource Pool Manager*. When extra resources are required, the *Resource Pool Manager* decides which resource pool to be used. We extended the existing MapReduce Service of Aneka to enable its interaction with the Provisioning Service, enabling the MapReduce scheduler to take advantage of public Cloud resources.

Data Exchange

To enable Aneka MapReduce to work across local and remote resources, a number of changes in the original MapReduce service were required. For example, we replaced the technology used for data exchange between nodes from Windows File Shares by HTTP. We made this decision because we observed that traffic generated by Windows File Shares is generally blocked by firewalls from companies and Universities. There were two options we could have adopted to overcome this problem: creating a VPN tunnel through the firewall, hiding the protocol form the firewall, or using a different protocol that is not generally blocked. The latter option was chosen not only because it did not add to the complexity of the network but also because it removes the tight coupling of the Aneka system with the Win32 API. Since Aneka is typically deployed in Windows-based Clusters, our current implementation adopts Internet Information Services (IIS) to serve the intermediary data files.

Remote Storage

Another change we developed concerns the storage layer of the MapReduce worker nodes. Every time a MapReduce worker starts up and registers with the master node, it reports which input data files it has available on its local storage. In the context of a Cloud provider such as Amazon Elastic Compute Cloud (EC2), this would translate to data files being stored as part of the disk image from which instances are created. This is undesirable for a number of reasons, for example a new image must be created each time the data files are changed. It also imposes a limit on the amount of data that is directly available to the worker nodes on EC2. Instead, Aneka's MapReduce Service was changed to use Amazon Simple Storage Service (S3) as the source of local input data files when running on EC2. Hence, when a MapReduce worker starts up on EC2, it reads a text file form S3 that lists the data files that are available on S3. When the worker registers with the master node, it reports as having those files locally available. As a Map task is assigned to a worker on EC2, the worker copies the input data file from S3 to its local disk before starting the actual processing of the task.

Intermediate Data Buckets and Reduce Tasks

Other MapReduce systems, such as Hadoop, define the number of Reduce tasks at the beginning of the application execution. As Map tasks produce key-value data, Hadoop hashes these key-values into buckets, where each bucket corresponds to a Reduce task. The Hadoop documentation[2] recommends that the number of Reduce tasks should be 0.95 or 1.95 times the number of nodes. However, as in our implementation additional resources can be provisioned during the Map phase, the final number of nodes present for the Reduce phase is not known at application submission time, and thus such limitations on number of Reduce tasks are not present.

Given the dynamic number of resources the Aneka MapReduce implementation generates, the initial service was also modified in order to separate the number of buckets used for hashing intermediate data from the number of Reduce tasks. With this change, the number of Reduce tasks is decided by the scheduler after the Map phase completes. The scheduler is also able to designate the hash buckets that a particular Reduce task will process.

VI. Performance Evaluation

In this section, we present experiments aiming at evaluating our policy for dynamic provisioning of resources for MapReduce applications with soft deadlines. We detail the environment where the prototype system was deployed, describe the deployed application, and present the results for different performance metrics.

A. Experimental Testbed and Sample Application

The experimental testbed used in the experiments presented in this paper is depicted in Figure 2. The environment is a hybrid Cloud composed of a local Cluster and a public Cloud. The local Cluster is composed of four IBM System X3200 M3 servers running Citrix XenServer. Each of these servers hosted two Windows 2008 virtual machines. Therefore, the local Cluster comprises eight worker nodes. The disk image for each of the VMs was stored on the local disk of the server hosting the virtual machine. Each virtual machine had access to two physical CPU cores, each one a Xeon 2.8 MHz, and 1.5 GB of RAM memory. The master node is a Dell Optiplex 745, with an Intel Core 2 Duo 1.6 MHz and 2 GB of RAM. All the machines were interconnected via a Gigabit Ethernet network.

Public Cloud resources were provisioned from Amazon EC2, USA East Coast data center, using *m1.small* instances, which have a single core equivalent to a 1.0-1.2 GHz CPU 2007 Xeon Processor[3], and 1.7 GB of memory, at the cost of US$0.085 per instance per hour.

The MapReduce application executed for the purpose of this evaluation is a word count application. This application contains a number of configurable parameters that affect the characteristics of the application, which allowed us to observe

[2]http://hadoop.apache.org/common/docs/r0.15.2/mapred_tutorial.html

[3]As at the time that this experiment was executed, according to http://aws.amazon.com/ec2/instance-types/.

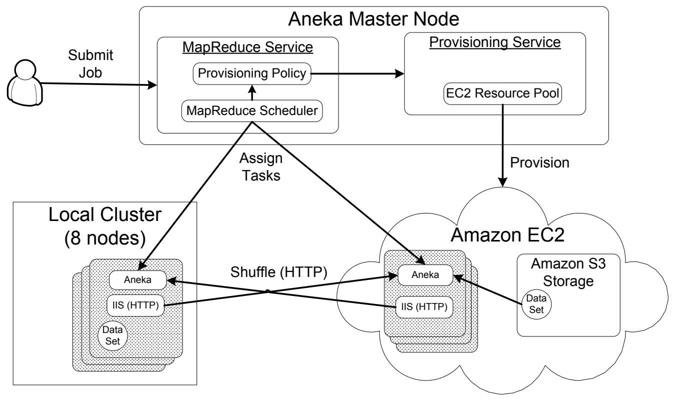


Fig. 2. The hybrid Cloud environment used in the performance evaluation.

the system under different conditions. The application can be configured to slow down each Map tasks by inserting a sleep statement. When such configuration option is utilized, the task performs the word count over one of the input files and then sleeps for the defined amount of time. In this case, the execution time of each Map task is the sum of the sleep time and the actual processing time.

Map tasks can aggregate their own word counts and only emit one count value for each unique word. This behavior can be changed with another configuration parameter, which dictates the maximum number for the count, meaning that words that occurred often would emit multiple key-value pairs. At the extreme, Map tasks will emit every individual word they encountered with a count value of one. Varying such maximum count per word allowed us to vary the volume of intermediary data. Reduce tasks aggregate counts for each word generated by difference Map tasks (keys) and sum the corresponding values to generate the final count for the word.

The dataset used for this application came from the Gutenberg project[4], which provides over 39,000 free books for download. A number of books were downloaded as text files and the metadata for each one was removed. These text files were then converted into 150 chunk files, where each file was just over 30 MB. Each Map task processes one of these chunk files. This resulted in a total dataset size of approximately 4.5 GB, which was copied to the local Cluster as well as to Amazon's S3 storage service.

The configuration parameters of our proposed provisioning policy were set as follows: *MARGIN* is 1 minute; *LOCAL_FACTOR* is 1.05; *REMOTE_FACTOR* is 1.15, to reflect the bigger variance in performance expected from public Cloud resources; and BOOT_TIME: is 6 minutes.

B. Performance Analysis

The two phases of a MapReduce application are investigated in separate experiments so that the conditions for each phase could be controlled independently, as different parameters are relevant on each of them. During the Map phase, the key parameters are the deadline for the Map phase and the

[4]http://www.gutenberg.org/

execution time of each task, whereas for the Reduce phase the key parameters are the size of the intermediate data generated by the Map tasks, what resources are used for Reduce tasks, and the sleep time inserted in each task.

Map Phase: The experiments executed for evaluation of the Map phase aimed at establishing if the provisioning policy can react accordingly to observed execution time and deadline and dynamically provision resources in an appropriate number in order to meet the deadline. Each set of experiments evaluating the Map phase was composed of 150 Map tasks.

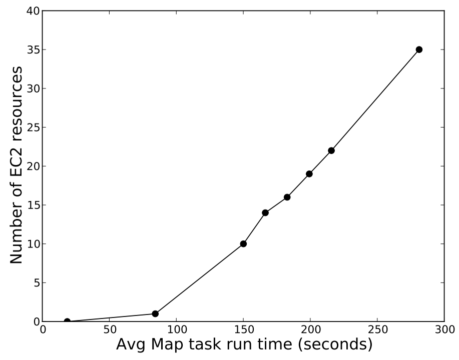
In the first set of experiments, we sequentially executed several requests for the word count application. On each request, we modified the sleep time in each Map task, and kept the deadline for completing the Map phase constant and equal to 30 minutes for each application.

Figure 3(a) shows the average execution time of Map tasks versus the number of resources that were provisioned on the public Cloud. As expected, we observed a linear relationship between task execution time and number of provisioned resources. This is because, as the Map tasks become longer and the deadline is not modified, more resources had to be provisioned in order to meet the deadline. In the first point in the graphs, the Map tasks were not delayed (sleep time set to 0) and no additional resources were required. Figure 3(b) shows the remaining time between the completion of the Map phase and the application deadline. Except for the first point, which did not need additional resources, our provisioning policy was able to successfully complete execution of the Map before the deadline. In all the cases, completion happens within five minutes of the deadline. The sparing time for deadline reflects the conservative policy we adopted in the experiments regarding boot time of Cloud resources (6 minutes, well above typical boot time of such resources) and performance variability of public Cloud resources (which we assumed 15%).

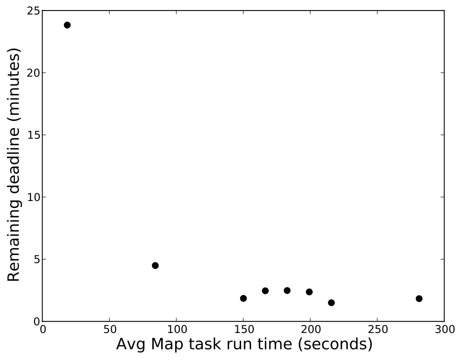
This indicates that our policy also accomplishes the requirements of minimizing the cost of dynamic provisioning by deploying as few machines in public Clouds as possible to meet the deadline.

The second set of experiments aimed at examining how the provisioning policy reacts to variation in deadlines while the execution time of Map tasks is kept constant. A two minute sleep time for each Map task was chosen as this permitted deadlines from 19 minutes up to an hour to be explored. All other settings were the same as on the previous experiments.

It was expected that the number of resources being provisioned to be inversely proportional to the deadline. Because a constant amount of work is being divided by the available time, a tighter deadline results in more resources to be required. The results showed in Figure 4(a) confirm such expectations. Figure 4(b) shows the remaining time between the completion of the Map phase and the application deadline. As in the previous experiment, all the completion times are within five minutes of the deadline, except where no dynamic provisioning was needed. This once again confirms that our policy minimizes the cost of dynamic provisioning by deploying as few machines in public Clouds as possible to meet the deadline.



(a)



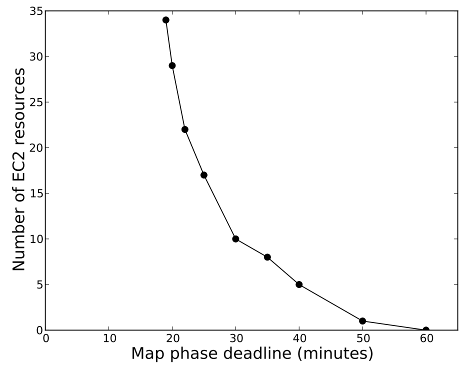
(b)

Fig. 3. Dynamic provisioning results with variable Map task execution times and fixed deadline (30 min). (a) Number of public Cloud resources provisioned (b) Remaining time before Map phase deadline.

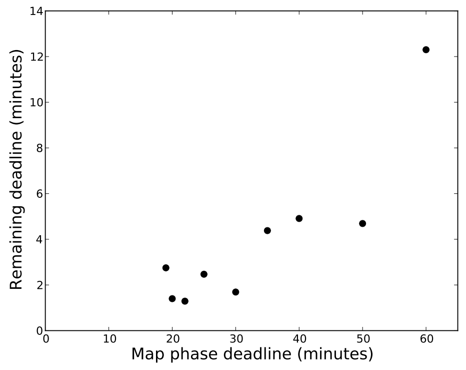
Reduce Phase: During the Reduce phase, the intermediate data generated by Map tasks is processed to generate the final output. Therefore, each Reduce task needs to collect all the intermediate data it has to process from the nodes where this data was generated by Map tasks. Afterwards, the data is sorted based on the keys, in order to enable Reduce tasks to process all the values for a given key.

For the experiments aiming at evaluating the performance of the Reduce phase, 20 EC2 m1.small instances were provisioned in the same location of previous experiments. These instances were already instantiated when the MapReduce application was submitted, because the focus of experiments for the Reduce tasks is not the provisioning of resources. By having the public Cloud resources available from the beginning of the application execution, we avoided interference in the results caused by resources unavailable at the beginning of the Reduce phase.

The Reduce experiments used two different settings for the Map tasks, which created different intermediate dataset sizes. The first setting had each Map task aggregating its own word counts, in such a way that only a single count value was emitted for each unique word. This resulted in the dataset of 352 MB referred as *Small* in this section. The second setting had a maximum count value of five. Therefore, count values for words were emitted at every 5th occurrence of the word plus one emission for the remainder. Thus, for example, a word

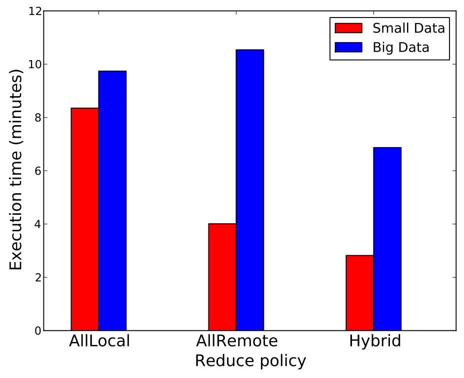


(a)

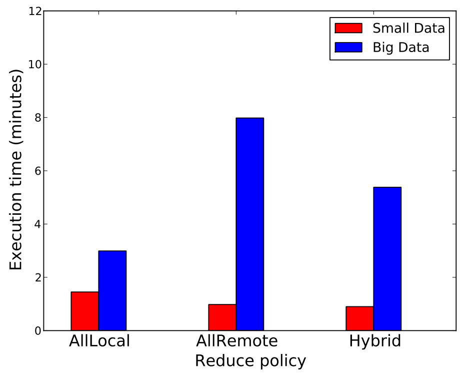


(b)

Fig. 4. Dynamic provisioning results with fixed Map execution time for each Map task and variable deadlines. (a) Number of public Cloud resources provisioned (b) Remaining time before Map phase deadline.



(a)



(b)

Fig. 5. Execution times of Reduce tasks with two different delays added to each Reduce tasks. (a) 1 min delay (b) 5 s delay.

observed eight times by a Map task results in two key-value pairs being emitted, one with the count 5 and the second with the count 3. This setting produced the dataset we refer as *Big* in this section, of 3422 MB of intermediate data to be reduced.

Similarly to the case of the Map phase, a sleep was inserted in the Reduce tasks in order to increase their execution time and observe how different sizes of Reduce tasks affect execution time of applications. Two different values for sleep, 5 seconds and 60 seconds, were used in the experiments. The three different Reduce policies were evaluated accordingly to the above methodology. For each policy, we measured the execution time of tasks with each considered sleep time, once for each dataset. Finally, we analyze only the time for actual execution of the Reduce phase, from the time the Map phase finishes to the time the last Reduce task completes. Therefore, time for download of results is disregarded in the experiments with public and hybrid Clouds.

Figure 5 shows that the results for Small and Big data are much closer for the *All Local* policy than for the other two policies, indicating that the time needed to process each task is more significant than the time required to transmit the data between the nodes. Compared to the other policies with the Small dataset, *All Local* has the worst performance, what helps in highlighting the benefits of using additional Cloud resources even during the Reduce phase. Results also show that *Hybrid* is the fastest policy with the Small dataset.

The analysis of the results for the Big dataset shows that the benefit of more resources is not as straightforward for large amounts of data. In particular, the *All Remote* policy, which uses 20 public Cloud resources, is significantly slower than the *All Local* policy, which uses eight local virtual machines. This can be attributed to an asymmetrical network performance, where sending data from the local Cluster to the EC2 instances was significantly slower than in the opposite direction. Hence, even though there were over twice as many resources available in the public Cloud, the time to transmit the data was detrimental to the policy. However, when the local and remote resources were used together by *Hybrid*, there was an improvement, because the remote resources were able to process part of the tasks while the local machines were in use.

Notice also that in Figure 5(b) Reduce tasks were slowed down by five seconds, whereas they were slowed down by 60 seconds in Figure 5(a). Even though it represents a twelve times slow down, the difference in the execution time of the whole Reduce phase delays varied from around 0.3 times (Big dataset, *All Remote*) to 8 times (Small dataset, *All Local*). This shows that the data transfer time dominates processing times for this type of application. Hence, with the Big dataset, *All Local* is the preferred policy as no data needs to be sent over the network. In the case of the Small dataset, however, the increased parallelism is still of benefit and *Hybrid* is the fastest approach for the scenario.

A comparison between Figure 5(a) and Figure 5(b) makes clear that the reduction of the execution times of each Reduce task had a bigger impact when the Small dataset was generated or when the *All Local* policy was used. This highlights that the Big dataset and the *All Remote* and *Hybrid* policies are strongly impacted by latency of long distance network connections between the local Cluster and the public Cloud.

Nevertheless, the experiments suggest that our approach for dynamic provisioning of Cloud resources in order to meet soft deadlines based on the execution time of the Map phase of MapReduce applications is effective in meeting deadlines while keeping the budget for execution as small as possible.

VII. Conclusions and Future Work

As MapReduce is becoming the prevalent programming model for building data processing applications in Clouds, the need for timely execution of such applications becomes a necessity. Because current approaches for deadline-aware execution of MapReduce applications in Clouds operate via admission control or prioritization of tasks, we proposed a different approach, where deadline of MapReduce applications is achieved with dynamic provisioning of public Cloud resources to complement local resources.

We presented a dynamic provisioning policy for MapReduce applications and a prototype implementation of the correspondent system in the Aneka Cloud Platform. Results showed that our approach, even though its lower complexity, delivers good results. Our policy was able to meet deadlines of applications, which are defined in terms of completion time of the Map phase, for increasing execution times of Map tasks and decreasing deadlines. Results showed that the connectivity between local and remote resources can be the solely dominant factor affecting the completion time of the whole application if the Reduce tasks are not computationally intensive.

We plan to extend our policy to optimize the provisioning for more complex scenarios, such as multiple independent applications (in order to minimize the total number of resources deployed for all applications) and composite MapReduce applications, where one application consumes the output of a previous application. We will also develop techniques for compensating for performance fluctuation of public Cloud instances and for straggle tasks, if such variation in performance is detected during application execution. We will also investigate techniques for unbalanced hashing and dynamic rebalancing when more buckets than Reduce tasks are present at the end of the Map phase. In the same direction, we will investigate partition merging and the optimal number of nodes for MapReduce implementations that overlaps the shuffle and data transfer with the Map phase.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: A software platform for .NET-based cloud computing," in *High Speed and Large Scale Scientific Computing*, W. Gentzsch, L. Grandinetti, and G. Joubert, Eds. Amsterdam, The Netherlands: IOS Press, 2009, pp. 267–295.

[4] Z. Fadika and M. Govindaraju, "LEMO-MR: Low overhead and elastic MapReduce implementation optimized for memory and CPU-intensive applications," in *Proceedings of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom'10)*, Indianapolis, USA, May 2010, pp. 1–8.

[5] Y. Geng, S. Chen, Y. Wu, R. Wu, G. Yang, and W. Zheng, "Location-aware MapReduce in virtual cloud," in *Proceedings of the 40th International Conference on Parallel Processing (ICPP'11)*, Taipei, Taiwan, Sep. 2011, pp. 275–284.

[6] T. Sandholm and K. Lai, "MapReduce optimization using regulated dynamic prioritization," in *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'09)*, Seattle, USA, Jun. 2009, pp. 299–310.

[7] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: Combining MapReduce and virtualization on distributed resources for bioinformatics applications," in *Proceedings of the IEEE Fourth International Conference on eScience (eScience'08)*, Indianapolis, USA, Dec. 2008, pp. 222–229.

[8] J. Polo, D. Carrera, Y. Bacerra, V. Beltran, J. Torres, and E. Ayguadé, "Performance management of accelerated MapReduce workloads in heterogeneous clusters," in *Proceedings of the 39th International Conference on Parallel Processing (ICPP'10)*, San Diego, USA, Sep. 2010, pp. 653–662.

[9] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, and W. W. Li, "A hierarchical framework for cross-domain MapReduce execution," in *Proceedings of the 2nd International Workshop on Emerging computational methods for the life sciences (ECMLS'11)*, San Jose, USA, Jun. 2011, pp. 15–22.

[10] Z. Fadika, E. Dede, J. Hartog, and M. Govindaraju, "MARLA: MapReduce for heterogeneous clusters," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*, Ottawa, Canada, May 2012, pp. 49–56.

[11] W.-T. Tsai, P. Zhong, J. Elston, X. Bai, and Y. Chen, "Service replication with MapReduce in clouds," in *Proceedings of the 10th International Symposium on Autonomous Decentralized System (ISADS'11)*, Kobe, Japan, Jun. 2011, pp. 381–388.

[12] F. Tian and K. Chen, "Towards optimal resource provisioning for running MapReduce programs in public clouds," in *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD'11)*, Washington DC, USA, Jul. 2011, pp. 155–162.

[13] A. Verma, L. Cherkasova, and R. Campbell, "Resource provisioning framework for MapReduce jobs with performance goals," in *Middleware 2011*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 7049, pp. 165–186.

[14] N. B. Rizvandi, J. Taheri, A. Y. Zomaya, and R. Moraveji, "A study on using uncertain time series matching algorithms in map-reduce applications," The University of Sydney, Technical Report TR672, 2011.

[15] S. Sehgal, M. Erdelyi, A. Merzky, and S. Jha, "Understanding application-level interoperability: Scaling-out MapReduce over high-performance grids and clouds," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 590–599, May 2011.

[16] X. Dong, Y. Wang, and H. Liao, "Scheduling mixed real-time and non-real-time applications in MapReduce environment," in *Proceedings of the 17th International Conference on Parallel and Distributed Systems (ICPADS'11)*, Tainan, Taiwan, Dec. 2011, pp. 9–16.

[17] K. Kc and K. Anyanwu, "Scheduling Hadoop jobs to meet deadlines," in *Proceedings of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom'10)*, Indianapolis, USA, May 2010, pp. 388–392.

[18] C. Jin and R. Buyya, "MapReduce programming model for .NET-based cloud computing," in *Proceedings of the 15th International European Parallel Computing Conference (EuroPar'09)*, Delft, The Netherlands, Aug. 2009, pp. 417–428.

[19] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, June 2012.