Poster: A Framework for Data-Intensive Computing with Cloud Bursting

Tekin Bicer
Department of Computer Science and Engineering
Ohio State University
bicer@cse.ohio-state.edu

David Chiu
School of Engineering and Computer Science
Washington State University
david.chiu@wsu.edu

Gagan Agrawal
Department of Computer Science and Engineering
Ohio State University
agrawal@cse.ohio-state.edu

ABSTRACT

In this work, we consider the challenge of data analysis in a scenario where data is stored across a local cluster and cloud resources. We describe a software framework to enable data-intensive computing with cloud bursting, i.e., using a combination of compute resources from a local cluster and a cloud environment to perform Map-Reduce type processing on a data set that is geographically distributed. Our evaluation with three applications shows that data-intensive computing with cloud bursting is feasible and scalable. Particularly, as compared to a situation where the data set is stored at one location and processed using resources at that end, the average slowdown of our system (using distributed but the same aggregate number of compute resources), is only 15.55%. Thus, the overheads due to global reduction, remote data retrieval, and potential load imbalance are quite manageable. Our system scales with an average speedup of 81% when the number of compute resources is doubled.

Categories and Subject Descriptors: H.3.4 [Information Storage and Retrieval]: Systems and Software – Distributed Systems

General Terms: Design, Performance.

Keywords: Cloud Bursting, MapReduce, AWS, S3.

1. INTRODUCTION

For many organizations, one attractive use of cloud resources can be through what is being referred to as *cloud bursting* or the *hybrid cloud*. These are scenarios where an organization acquires and manages in-house resources to meet its base need, but can also harness additional resources from a cloud provider to maintain an acceptable response time during workload peaks. Cloud bursting can be an attractive model for organizations with a significant need for HPC. But despite the interest in HPC on clouds, such organizations can be expected to continue to invest in in-house HPC resources, with considerations such providing best performance, "security" needs of certain applications, and/or desire for having more control over the resources.

At the same time, through cloud bursting, organizations can also avoid over-provisioning of base resources, while still providing users better response time. In fact, it is quite well documented that users routinely experience long delays while accessing resources from supercomputing centers. As one data point, in 2007, the ratio between wait time and execution time was nearly 4 for the Jaguar supercomputer at Oak Ridge National Lab (ORNL). Besides the need for reducing wait times for user satisfaction and productivity, another consideration is of *urgent high-end computations*, where certain compute-intensive applications rely on rapid response.

Cloud bursting has so far been associated with the use of additional computing resources from a cloud provider for applications. We do not believe that it needs to be limited in this fashion. As next generation applications are expected to see orders of magnitude increase in data set sizes, cloud resources can be used to store additional data after local resources are exhausted. While the available bandwidth to cloud-based storage is quite limited today, ongoing developments (such as building dedicated high speed connections between certain organizations and a cloud provider) are addressing this issue. Thus, one can expect efficient storage and access to data on cloud resources in the future.

In this work, we consider the challenge of data analysis in a scenario where data is stored across local resource(s) and cloud resources. Analysis of large-scale data, or data-intensive computing has been a topic of much interest in recent years. Of particular interest has been developing data-intensive applications using a high-level API, primarily, Map-Reduce framework, or its variants. Map-Reduce has interested cloud providers as well, with services like Amazon Elastic MapReduce now being offered.

This work describes a middleware that supports Map-Reduce type API in an environment where the data could be split between a local cluster and a cloud resource. Data processing is then performed using computing resources at both ends. However, to minimize the overall execution time, we allow for the possibility that the data at one end is processed using computing resources at another end, i.e., work stealing. Our middleware considers the rate of processing together with distribution of data to decide on the optimal processing of data.

2. MIDDLEWARE DESIGN

Our proposed software framework can be viewed as an implementation of Map-Reduce that can support the *transparent remote data analysis paradigm*. In this paradigm, analysis of data is specified with a high-level API (Map-Reduce or its variant), but the set of resources for hosting the data and/or processing it are geographically distributed.

Figure 1 illustrates the execution paradigm facilitated by the middleware. The *head* node is responsible for inter-cluster communication and schedules jobs to be executed between clusters. Each cluster is managed by its own *master* node, which communicates directly with the *head* node and distributes the jobs to its *slaves*. The actual work is performed on the *slaves*, which retrieve and process the data.

Whenever a cluster's job pool diminishes, its corresponding *mas-*

SC'11 Companion, November 12–18, 2011, Seattle, Washington, USA.
ACM 978-1-4503-1030-7/11/11.

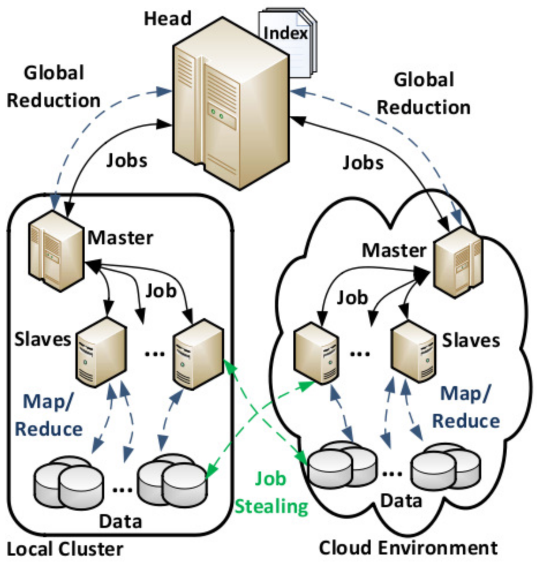


Figure 1: Middleware for Data Processing on Hybrid Clouds

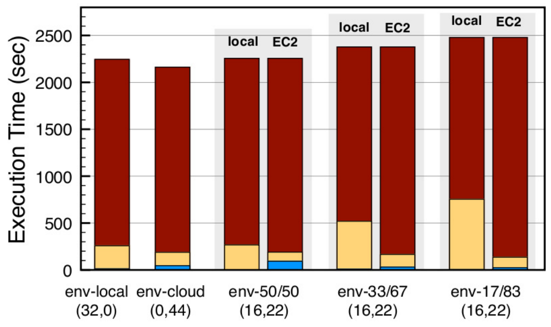
ter requests jobs from the *head* node. The *master* then assigns a group of jobs to the cluster based on data locality, e.g., if there are locally available jobs in the cluster, then those will be assigned first. Once all of the local jobs are processed, the remote jobs are selected from files which the minimum number of nodes are processing to reduce contention. Remote job processing is shown as "job stealing" in the figure. After all the jobs are processed, the *head* node enters the global reduction phase by requesting and combining the locally reduced data and forming the final result.

The job assignments in our system include the metadata information of the data chunks. Metadata information of a data chunk consists of location, offset, and size of each unit data. When a job is assigned to a *slave*, it retrieves the data chunk according to the given metadata information. If the data chunk is locally available, continuous read operations are performed. However, if the data chunk needs to be retrieved from a remote location, i.e. job stealing, multiple retrieval threads are used to utilize the available bandwidth. The processing of the data chunk begins at the slaves following data retrieval.

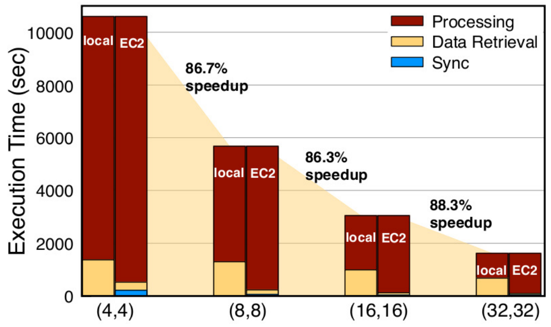
Load balancing is maintained through the *slaves*' on-demand job request scheme. Clearly, the *slave* nodes that have higher throughput (e.g., faster compute instances inside a cloud cluster) are expected to process more jobs. In similar fashion, a *master* node also requests a group of jobs from the *head* on demand, thus ensuring that the clusters with more computational throughput would perform more processing.

3. EXPERIMENTAL RESULTS

We execute kmeans application over five configurations. These configurations involve the same aggregate computing power. In the first two configurations, which are `local` and `cloud`, the computing resources and the datasets are at the same location. In other words, these two configurations involve centralized storage and processing, and are used as the baseline. The next three configurations involve a 50-50 split of computing power across local and cloud resources. Moreover, within these three configurations, there is a varying amount of *skew* or *unevenness* in the distribution of data. The data distribution for `env-33/67` is 33% (40GB) of the data is hosted locally, while 67% (80GB) is being hosted in Amazon S3. By varying the amount of data skew, we increase the amount of remote data retrieval that may be needed, and thus can observe its impact on the overall performance. The number of cores is empirically determined according to the computational power they provide. We set the throughput power of each cluster as close as possible and evaluated the overhead of usage of the cloud with the local resources.



(a) Cluster Exec. Time



(b) System Scalability

Figure 2: Cloud Bursting with KMeans Application

In Figure 2(a), we can see that this application is dominated by computation. Even high fraction of the data is stored in S3, the computation time takes longer than the data retrieval time in local cluster. The displacement of stored data from local clusters to S3 is identical with moving locally available jobs to the cloud. Therefore, the local cluster finishes its local jobs sooner and fetches more from S3, which results in higher retrieval times on local cluster. The overheads of the hybrid configurations are quite low. In fact, `env-17/83` is still performing at around 90% efficiency of `env-local`.

In Figure 2(b), we show the scalability results of `kmeans`. Because `kmeans` is compute-intensive and the data set is fairly large, the overall computation time is quite high. The synchronization overhead in this set of experiment ranges from 0.1% to 2.5% and the maximum synchronization overhead is seen in the $(4, 4)$ configuration on cloud cluster. The reason is that small number of cores results in longer job processing times and the idle time of the clusters increases.

We also repeated the same set of experiments with `pagerank` and `knn` applications. Our results with these applications are similar to those of kmeans application.