Brief Announcement: Modelling MapReduce for Optimal Execution in the Cloud

Alexander Wieder[†], Pramod Bhatotia[†], Ansley Post[†‡], and Rodrigo Rodrigues[†]
[†]Max Planck Institute for Software Systems (MPI-SWS) and [‡]Rice University
{awieder, bhatotia, abpost, rodrigor}@mpi-sws.org

ABSTRACT

We describe a model for MapReduce computations that can be used to optimize the increasingly complex choice of resources that cloud customers purchase.

Categories and Subject Descriptors

D.4.0 [**Software**]: Operating Systems—*General*

General Terms

Design, Performance, Management

1. INTRODUCTION

Cloud services are being increasingly used to outsource computations. But as cloud services become more popular, the variety of services that are offered is becoming overwhelming. In this research we aim at providing a model based on linear programming for cloud computations that enables customers to make an optimal choice of which resources to allocate, e.g.., one that minimizes monetary costs. This can then be integrated into a system [3] that automates resource allocation.

This paper focuses on modelling a restricted type of computations, namely MapReduce [2] jobs. This is not only a relevant programming paradigm that is increasingly used for large-scale computational jobs, but will also offer us a starting point from which we can generalize our techniques.

2. THE SCHEDULING PROBLEM

A MapReduce job consists of two distinct phases, the *Map* phase and the *Reduce* phase, that are strictly processed in that order without overlap. Each phase is a distributed execution of parallelizable computations, which scales well with the number of processing nodes. To find an optimal schedule and resource allocation strategy for MapReduce, we first model each phase independently using dynamic linear programming, and later combine both phases to determine a globally optimal strategy.

2.1 Basic Model

Cloud computing providers offer various services with different pricing schemes and performance characteristics. We consider m distinct cloud services (e.g., Amazon EC2, Amazon S3), $F_1, ..., F_m$, that provide different types of resources (e.g., computing cycles, storage). Let T be the upper bound for the deadline to finish the computation in terms of number of time intervals, which are the granularity of the execution progress (e.g., of one hour each).

2.1.1 Storage Model

To execute the *Map* phase, the input data from the source storage has to be uploaded to a storage service in the cloud for processing. We model this upload in a time-step fashion for T intervals. For interval t, the source storage contains $source_t$ amount of data and $upload_{(i,t)}$ denotes the amount of data uploaded from the source storage to the storage service F_i. The total uploaded data expressed as $storeIn_{(i,t)}$ will be stored in F_i until the execution phase is finished. Data storage and upload is *flow preserving*, which we express by the following constraints:

$$\forall i,t: \; source_t - \sum_{i=1}^{m} upload_{(i,t)} = source_{t+1} \tag{1}$$

$$\forall i,t: \; storeIn_{(i,t-1)} + upload_{(i,t)} = storeIn_{(i,t)} \tag{2}$$

The available upload speed can be expressed in the model by adding a constraint that restricts the total amount of data in each time-step to the upload capacity.

2.1.2 Computation Model

We now extend our model to computations, in which the data uploaded to the cloud is processed, and whose result is stored in a storage service. We also model data processing in a time-step manner for T intervals. In interval t, the uploaded data $storeIn_{(i_1,t)}$ in storage service F_{i_1} can be processed by a computational service and then the result $storeOut_{(i_2,t)}$ is stored at F_{i_2}.

The amount of data that is processed in each time interval t is bounded by the number of computing nodes that we choose to run during that interval. Also, we can only process input data in the cloud that has already been uploaded. Let $proc_{(i,t)}$ denote the amount of data which is processed by cloud service F_i in time interval t. We can therefore represent the constraints for computations as follows:

$$\forall i,t: \; \sum proc_{(i,t)} \leq nodes_{(i,t)} \cdot capacity_i \tag{3}$$

$$\forall t: \; \sum_{t'=1}^{t} \sum_{i=1}^{m} proc_{(i,t')} \leq \sum_{i=1}^{m} storeIn_{(i,t)} \tag{4}$$

Here, $nodes_{(i,t)}$ denotes the number of computing nodes rented in interval t from computing service F_i, and $capacity_i$ denotes the processing capacity of a single node for F_i.

Copyright is held by the author/owner(s).
PODC'10, July 25–28, 2010, Zurich, Switzerland.
ACM 978-1-60558-888-9/10/07.

2.1.3 Execution Cost

The monetary cost of the *Map* phase can be expressed as the cumulative sum of the cost incurred in each interval over time T. For time interval t, the cost can be expressed as the sum of the cost incurred for uploading the data, processing the data and storing the result in cloud. We calculate the cost for each time interval based on the amount of resources consumed per cloud service. For instance, the computation cost in time interval t is the number of machine-hours used in this interval multiplied by the price per machine-hour. Formally, we express the total monetary cost over T as follows:

Let $y_{(i,t)}$ be the number of units of cloud service F_i purchased for time interval t, and let b_i be the price per unit for F_i. The total cost C for such a configuration is

$$C = \sum_{t=1}^{T} \sum_{i=1}^{m} (b_i \cdot y_{(i,t)}) \tag{5}$$

Note that this monetary cost, as well as other characteristics captured in our model such as execution time, can be used in the objective function for optimization. Since no negative amount of resources can be purchased, we automatically have the constraints $\forall i: \; y_{(i,t)} \geq 0$. This applies to all variables we used throughout this paper.

2.2 Dynamic Pricing

Recently, Amazon started offering spot market pricing, where customers bid the price they want on unused Amazon EC2 capacity, and the price tag reflects the current supply and demand. Furthermore, customers can use the history of spot prices as a predictor of their evolution, to develop a bidding strategy.

We thus extend our model to include dynamic pricing in spot markets. Given our model where computations are divided into discrete time-steps, spot prices can easily be incorporated by setting the price in each time-step to an estimated spot price. These estimates could potentially be derived by extrapolating past pricing patterns. Let $E[b_{(i,t)}]$ denote the estimated price per unit of cloud service F_i for time interval t. Thus, the modified total cost C' can be expressed as follows:

$$C' = \sum_{t=1}^{T} \sum_{i=1}^{m} (E[b_{(i,t)}] \cdot y_{(i,t)}) \tag{6}$$

2.3 Data Migration

Since we consider multiple storage services in our model, we may choose to migrate data between them during the execution. We include migration by adding transitions in each time-step t from $storeIn_{(i_1,t)}$ to $storeIn_{(i_2,t+1)}$. These transitions express migrating input data from the storage services F_{i_1} to F_{i_2}. Similarly, we add transitions for migrating the output data *storeOut* in each time-step. Note that the transitions for data migration go from one time-step t to the next one $t+1$, rather than staying within the same time step. This allows us to express that data migration is not completed instantly. The cost for migration can be added to the storage cost per time-step.

2.4 Combining Phases

So far we only considered the *Map* phase in our model. The *Reduce* phase can be modelled in a similar manner, except for the fact that we do not include a source storage or data upload – the *Reduce* phase takes the result of the *Map* phase as the input. Hence, in our model in each time-step t we add transitions from the output storage of the *Map* phase $storeOut_{(i,t)}$ to the input storage of the *Reduce* phase $storeIn_{(i,t+1)}$.

We enforce that these two phases cannot overlap by specifying that the amount of data flowing to the next phase has either to be 0 or the full output data. We specify this property as a linear programming constraint using a *semi-continuous* variable, that can hold either 0 or the full output data size. After combining the two phases of the job, we model the download of the final result from the output storage of the *Reduce* phase by adding transitions to the destination storage.

2.5 Resource Overlap

In our previous explanation of the model we have assumed that each service provides only a single type of resource, either storage or computation. However, services can provide both of them (and potentially other types) simultaneously in practice. For instance, we can opportunistically store data on the virtual disk of running VMs, leveraging this spare resource at a low extra cost.

Our model accommodates this easily, since it distinguishes cloud services from the resources they provide. Thus, in addition to the pricing and performance characteristics we already specify for each cloud service $F_1, \ldots, F_m$, we also specify the quantities of other resources $R_1, \ldots, R_n$ each of the services offers. For instance, in this model a pure storage service like Amazon's S3 will provide only storage resources while instances of Amazon's EC2 service provide both computation and storage resources.

3. CONCLUSIONS

This paper reports on our experience in modelling the choice of cloud resources that can be acquired to execute MapReduce computations. We first present a basic model that includes storage and computation resources; then we refine it to reflect several alternatives that are presented to cloud customers.

This exercise is part of a larger effort of building systems that aid users in making better use of cloud resources. As such, it opens several avenues of future work, such as building systems that use this model [3], generalizing the current model to other types of computational jobs that can be outsourced, or allowing the model itself to be inferred automatically from sampling an actual execution of the computation.

4. REFERENCES

[1] Amazon Spot Instances. `http://aws.amazon.com/ec2/spot-instances/`

[2] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proc. of the Sixth Symposium on Operating Systems Design and Implementation (OSDI '04)*.

[3] Wieder, A., Bhatotia, P., Post, A., and Rodrigues, R. Conductor: Orchestrating the clouds. Under submission. Draft available at `http://www.mpi-sws.org/~bhatotia/Conductor.pdf`