See Spot Run: Using Spot Instances for MapReduce Workflows

Navraj Chohan[†*] Claris Castillo Mike Spreitzer Malgorzata Steinder
Asser Tantawi Chandra Krintz[†]
IBM Watson Research
Hawthorne, New York

[†]*Computer Science Department*
University of California, Santa Barbara, CA

Abstract

MapReduce is a scalable and fault tolerant framework, patented by Google, for computing embarrassingly parallel reductions. Hadoop is an open-source implementation of Google MapReduce that is made available as a web service to cloud users by the Amazon Web Services (AWS) cloud computing infrastructure. Amazon Spot Instances (SIs) provide an inexpensive yet transient and market-based option to purchasing virtualized instances for execution in AWS. As opposed to manually controlling when an instance is terminated, SI termination can also occur automatically as a function of the market price and maximum user bid price. We find that we can significantly improve the runtime of MapReduce jobs in our benchmarks by using SIs as accelerators. However, we also find that SI termination due to budget constraints during the job can have adverse affects on the runtime and may cause the user to overpay for their job. We describe new techniques that help reduce such effects.

1 Introduction

MapReduce is a general computational model that originated from the functional programming paradigm for processing very large data sets in parallel. A scalable, fault tolerant approach of MapReduce has been popularized and recently patented by Google [5, 6]. This implementation operates on data in the form of key/value pairs and simplifies how large-scale data reductions are expressed by programmers. The system automatically partitions the input data, distributes computations across large compute clusters, and handles hardware and software faults throughout the process. Since the emergence, use, and popularity of MapReduce for a wide range of problems, many other implementations of the process have emerged. The most popular of which is Hadoop [7], an open-source implementation of Google MapReduce. Hadoop is currently in use by Yahoo!, Facebook, and Amazon, among other companies.

Given its ease of use and amenability to parallel processing, MapReduce is employed in many different ways within cloud computing frameworks. Google employs its MapReduce system for data manipulation within its private compute cloud and AppScale, the open-source implementation of the Google App Engine (GAE) cloud platform, exports Hadoop Streaming support to GAE applications [3]. The Amazon Web Services cloud infrastructure makes Hadoop and Hadoop Streaming available as a web service called Elastic Map Reduce [1].

In December of 2009, Amazon announced a new pricing model for AWS called Spot Instances (SIs). SIs are ephemeral virtual machine instances for which users pay for each completed runtime hour. A user defines a maximum bid price, which is the maximum the user is willing to pay for a given hour. The market price is determined by Amazon, which they claim is based on VM demand within their infrastructure.

If a VM is terminated by Amazon because the market price became higher or equal to the maximum bid price, the user does not pay for any partial hour. However, if the user terminates the VM, she will have to pay for the full hour. Furthermore, a user pays the market price at the time the VM was created, given that it survives the next hour. The cost of the hours that follow may differ depending on the market price at the start of each consecutive hour.

SIs are an alternative to on-demand and reserve VM instances in Amazon. On-demand instances have a set price for each hour that does not change. Reserve instances have a cheaper per-hour price than both on-demand instances and SIs, but the user must lease the VMs for long periods of time (1 or 3 year terms). SIs therefore provide inexpensive computational power at the cost of reliability (variable and unknown VM lifetime). The reliability is a function of the market price and the users maximum bid (limited by their hourly bud-

[*]Navraj Chohan pursued this research as an intern at IBM Research.

get).

In this work, we investigate the use of SIs for MapReduce tasks. SIs fit well into the MapReduce paradigm due to its fault tolerant features. We use SIs as accelerators of the MapReduce process and find that by doing so we can significantly speed up overall MapReduce time. We find that this speedup can exceed 200% for some workloads with an additional monetary cost of 42%.

However, since SIs are less reliable and prone to termination, faults can significantly impact overall completion time negatively depending on when the fault occurs. Our experiments experience a slow down of up to 27% compared to the non-SI case, and 50% compared to an accelerated system in which the fault does not occur.

Since the likelihood of termination is dependent on the market price of the VM and the user defined maximum bid price, we investigate the potential benefit and degradation (cost of termination) of using SIs for MapReduce given different prices. We also use the pricing history of Amazon SIs to determine how much to bid as well as how many machines to bid for. By using this characterization for a given bid and market price, we compute expected VM lifetimes for users. Such a tool enables users to best determine when to employ SIs for MapReduce jobs.

2 Background

We first briefly overview the Hadoop MapReduce process. Using Hadoop, users write two functions, a mapper and a reducer. The map phase takes as input a file from a distributed file system, called the Hadoop Distributed File System (HDFS), and assigns blocks (splits) of the file as key-value pairs to mappers throughout the cluster. HDFS employs replication of data for both fault tolerance and data locality for the mappers. Mappers (map tasks) consume splits and produce intermediate key-value pairs which the system sorts and makes available to the reducers. Reducers (reduce tasks) consume all pairs for a particular key and perform the reduction. Reducers then store the resulting output to HDFS. The result may be a final computation or may itself be an intermediate set of values for another MapReduce tuple.

Each machine is configured with a maximum number of mapper and reducer tasks slots. The number of slots depends upon the resources available (i.e. number of CPU cores and memory) as well as the type of job being run (CPU-bound versus IO-bound). The master runs a Job-Tracker process which assigns work to available worker slots. Slave nodes run Task-Trackers which have their task slots assigned work as it becomes available by the Job-Tracker. Each Task-Tracker can run a custom configuration. It can be designated to run only mappers, only reducers, or, as is typical, some combination of the two.

Hadoop tolerates failures of a machine through the use of replication. Output data can be regenerated given there are live replicas of the input splits. The replication policy for Hadoop is rack-aware and will place one copy on the same physical rack and the second off-rack. Hadoop also tolerates bad records. Records which cause exceptions are retried a default of three times and then skipped to ensure the entire job is not halted due to a single bad record. This issue can come about when buggy third-party software is used.

Hadoop uses heart-beat messages to detect when a machine is no longer operable. Data which was lost due to a failure is replicated to ensure that the configured number of replicas exist.

The time for a MapReduce job in Hadoop is dependent on the longest running task. Tasks that are few in number and those that continue execution once most others have completed are called stragglers. The system can speculatively execute stragglers in parallel using idle task slots in an attempt to reduce time to completion. The authors in [11] provide details on the impact of stragglers in virtualized environments.

3 Data Analytic Cloud

In this section we describe what we envision to be a data analytic cloud which uses MapReduce for analyzing data and the cost associated with running such a service. The scenario which this paper focuses on relies on a provider to host their large data sets in a public cloud. The data is stored in a distributed file system running on a subset of leased VMs in the cloud. In addition, the provider may provision the data analytic engine required for processing or querying the data. In this paper we consider the MapReduce framework as that engine, although this work also carries to using higher level query languages such as Pig [9] and Hive [10]. Users submit MapReduce jobs, and the provider charges the user an hourly rate, along with the option to speedup their job at an additional cost. In order to maximize profit, the provider uses the cheapest source of computation available. Amazon's EC2 SI pricing is competitive in this area, being as low as 29% of the cost of an EC2 on-demand instance.

Amazon's Elastic MapReduce is another option available, giving users an easy and cost effective way to analyze large data sets. Data, at the time of writing this paper, is free to upload into their Simple Storage Service (S3) and free to transfer within EC2, but transferring out is $0.15 per GB and storage per month per GB is $0.15. A 1TB set of data cost $150 per month to store, and $150 per transfer out. There is also an additional cost for PUT and GET request for S3 at $0.01 per 1000

requests. Elastic MapReduce is spawned with a user defined number of instances. The user only pays for the number of VM hours used at an additional 17.6% charge of the on-demand VM instance price.

The minimum cost of hosting a 1TB data set in a Hadoop cluster (with a 3 year term) using just local instance disk space, with three times replication, costs $194 dollars a month (20 small instance VMs with 160GB each for a total of 3.2TB of distributed storage). The Elastic MapReduce service with S3 is more affordable if the total amount of cost for VM instances is less than $44 dollars a month, which affords 440 VM hours a month or 22 hours of processing for 20 small VM instances. The disadvantages of storing the data in S3 is that the MapReduce cluster loses the data locality a local HDFS cluster provides.

4 Analysis

We next investigate how best to employ Hadoop within a cloud infrastructure for which virtual machine instances are transient. Our goal is to investigate how best to do so given the Spot Instance (SI) option offered by Amazon Web Services (AWS). SIs offer a cost effective alternative to on-demand instances since the cost of their use is dependent on market-based supply and demand of instances. We find that SIs can be as low as 29% of the cost of on-demand instances. SIs trade off termination control for such cost savings. SIs are good for short running jobs that can tolerate termination, i.e. faults in the execution process. MapReduce is an ideal candidate for SIs since we can use additional nodes to accelerate the computation.

However, since the time to complete a MapReduce process is dependent upon how many faults it encounters, we must also consider SI termination. Since SI termination is dependent upon market price and maximum bid price, we are interested in using this information to estimate the likelihood of termination.

To enable this, we consider bid prices independent of market prices since there is very limited information available from Amazon as to how they determine the market price. Amazon does not reveal bids by users or the amount of demand. Table 1 shows the pricing of different instance types in the western US region. The SI pricing is an average of prices since they were first introduced in December of 2009 till March of 2010. The small instance type uses a 32-bit platform, while the rest are 64-bit. An EC2 compute unit is equivalent to a 1.0 to 1.2 GHz 2007 Xeon or Opteron processor [1].

4.1 Spot Instance Characterization

We model the SI lifetimes by building a Markov Chain with edges being the probability of price transitions for each hour interval. Given the transitional probabilities we can calculate n-step probability using a variant of the Chapman-Kolmogorov equation:

$$P(i,b,n) = \sum_{j \notin B} M_{ij} P(j,b,n-1) \tag{1}$$

where

$$p(i,b,0) = \begin{cases} 0 & \text{if } i \in B \\ 1 & \text{if } i \notin B \end{cases} \tag{2}$$

The starting market price at the time of VM creation is i and n is the number of time unit steps. The set of prices which are over the bid price, b, are in set B. M_{ij} is the probability matrix of a price point from i to j. Pricing history was collected over time using Amazon's EC2 tools and can be attained from [4]. $P(i,b,n)$ is solved recursively, where each step depends on the previous one. The base case is a binary function of whether or not the bid price is greater than the market price.

Figure 1 shows the probability of a VM running for n hours. The figure has different maximum bid prices given the market price being $0.035 at the time of starting the instance. As the maximum bid decreases, the probability of the SI staying up decreases as well. Some small increments in the bid price can give much larger returns in probabilities as can be seen when incrementing the bid price from $0.041 to $0.043, whereas other increments give very little return ($0.037 to $0.039). A SI in this case has more than an 80% chance of making it past the first hour given the market price was less than the bid price at the start of the VM. Bids that are less than or equal to the market price at the start of the VM would stay at 0% probability ($0.035 for example which is not viewable because it is directly overlaid on the x-axis).

Figure 2 has two sets of data for comparison. The data set labeled "A" is from mid-January 2010 to mid-March 2010, while the data labeled "B" is from mid-March 2010 to the end of May 2010. A comparison of the two models shows that the past pricing model is a good indicator of future pricing. It should be noted that data prior to mid-January was not used in building the model as shown in Figure 1 because as reported in [2] there was a bug in the pricing algorithm prior to this date which has since been fixed. The bug's impact can be seen in the pricing visualized in [4] where prices stabilized post January 15th. This explains the smaller range of prices between Figure 1 and Figure 2.

Using Equation 1, we can calculate the expected lifetime, l, of a VM given a starting market price, i, a given

Instance Type	Average Price	StDev	On-Demand Price	EC2 Compute Units	Memory (GB)	Storage (GB)
m1.small	$0.0399	0.001327	$0.095	1	1.7	160
c1.medium	$0.0798	0.002551	$0.190	5	1.7	350
m1.large	$0.1673	0.04163	$0.380	4	7.5	850
m2.xlarge	$0.2397	0.007489	$0.570	6.5	17.1	420
m1.xlarge	$0.3197	0.009045	$0.760	8	15	1690
c1.xlarge	$0.3233	0.02469	$0.760	20	7	1690
m2.2xlarge	$0.5593	0.01756	$1.340	13	34.2	850
m2.4xlarge	$1.1164	0.03288	$2.68	26	68.4	1690

Table 1: Prices of different VM instances from the US west region. Instances labeled with "m1" are standard instances, "m2" are high-memory instances, and "c1" are high-CPU instances. EC2 compute units are based on CPU cores and hardware threads. All instances here are for the Linux operating system. Costs are obtained from [1].

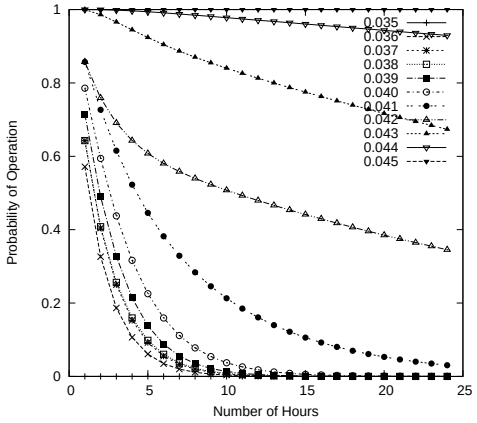


Figure 1: The probability of a small VM instance staying operational over time given a starting price of $0.035 with varying bids.

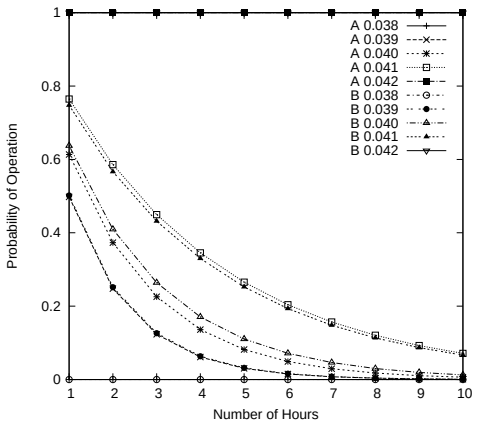


Figure 2: A comparison for verifying the pricing and lifetime model of a small VM instance given a starting price of $0.038 and varying bids.

bid, b, and a max run time of τ time units:

$$E(l) = \sum_{n=1}^{\tau} nP(i, b, n) \tag{3}$$

We can determine the amount of expected work a VM should achieve given the lifetime of a VM. This value can be used in the planning of backing up data and hence reduce the impact of failure. Moreover, it can be used in bidding strategies to ensure the greatest amount of SIs can be requested without fear of going over your maximum allocated budget.

4.2 Cost of Termination

We define the cost of a termination as the amount of time lost compared to having the set of machines stay up until completion of the job. The minimum cost is

$$\delta + (fM/s)/(s - f) \tag{4}$$

where the total time taken to complete the mappers is M. The total number of mappers is a function of block size and the size of the input file. The total number of slaves is s, the number of machines terminated is f, and the time spent waiting for a heart-beat timeout to occur while useful work could be done is δ. Early termination of a machine into the map phase allows for an overlap of when the termination is detected and the rest of the cluster is doing useful map work (i.e. no map slot goes idle). Work is equally divided given the machines are homogeneous.

Termination also results in the loss of reducer slots if that machine was configured so. This may or may not be an additional cost of failure depending on the job configuration which can specify the number of total reducers. This potential cost is not reflected in Equation 4 due

to its application specific and configuration specific nature. Termination after all the mappers have finished, sees the most expense of the fault detection, forcing a re-execution of all mappers completed on the terminated machine, even those which have been consumed by reducers and will not be consumed again.

4.3 Evaluation

Our initial experiment consists of five small-sized on-demand instances on EC2 with one node as the master, and four as slaves. The slaves were configured with two map slots and one reduce slot. Additional EC2 SIs which are added for speedup also have the same configuration. Each data point is an average of five trials. The applications are wordcount, pi estimation, and sort. Wordcount counts the occurrence of each word of an input file. Pi estimation uses a quasi-Monte Carlo procedure by generating points for a square with a circle superimposed within. The ratio of points inside versus outside the circle is used to calculate the estimation. Sort uses the MapReduce framework's identity functions to sort an input file.

Figure 3a has the speedup of each job with the number of SIs varied. The speedup is normalized to the original HDFS cluster configuration. Speedup is linear for all three applications. The price for speedup is in Figure 3b where each additional SI cost $0.04 per hour. Each job ran for less than one hour, therefore had the job been running for n hours, the y-axis would be a multiple of n.

Our second experiment was using five machines as the HDFS cluster, and one machine as an accelerator. Figure 4 has the speedup breakdown of adding an accelerator as well as the relative slowdown when the SI is terminated halfway through execution. The detection of machine faults was set to 30 seconds to minimize δ for these experiments, where the default is 10 minutes. The default delay is set sufficiently large for the purpose of distinguishing between node failure and temporary network partitioning and had our experiments used the default value δ would have grown accordingly.

For Figure 4 the mapper portion is from when the first mapper begins and the last mapper ends. The shuffle period is where map output is fetched by reducers. This phase runs in parallel with mappers until the last mapper output is fetched. Reducers fetch and merge-sort the data as map output becomes available. They finish merge sorting the remaining intermediate data and proceed to run the reduce procedure once the shuffle phase is complete. The reduce procedure does not start until all map output is accounted for.

As expected, we see speedup for all applications with the addition of an SI. Yet the cost of losing the accelerator actually slows down the application sufficiently, to the point where it was faster with the original setup. If the SI ran longer than an hour, it would have cost the user money with no work to show for it. On the other hand, if the SI was terminated before the first complete hour, no money is lost. The completion time is hampered in both cases. Section 5 presents the solution we are pursuing to alleviate this problem.

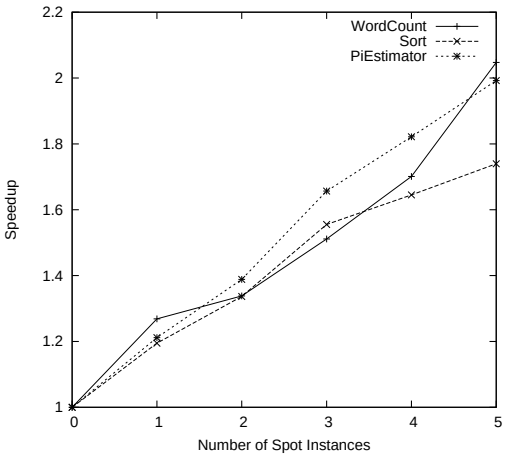
The addition of SIs improves the completion time of the mappers, but may not improve the completion time of the reduce phase. Many applications have a sole reducer at the end of the map phase because it requires a holistic view of the map output. Additionally, the runtime of the reducer is dependent on the amount of intermediate data generated. The amount of intermediate data is subject to the MapReduce application, and the input data. The use of a combiner also reduces the amount of intermediate data, which is invoked at the end of a mapper performing an aggregation of mapper output. Our wordcount benchmark uses a combiner which essentially does the same job as the reducer at a local level. The aggregated map output helps to decrease the reducer workload in both the amount of data which must be fetched and processed.

5 Discussion

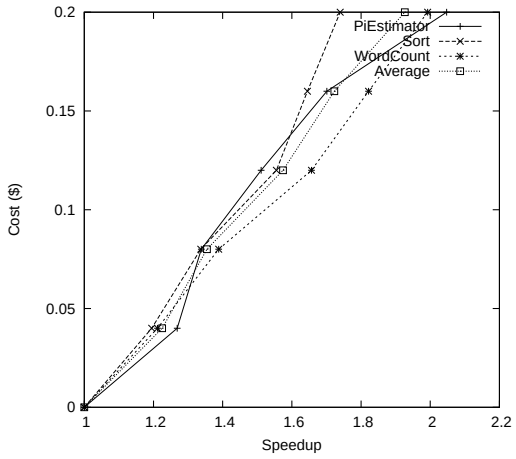
Had we kept adding SIs to the system in our first experiment, we would expect to get a diminishing return in the amount of speedup an application sees. For each SI, data must be streamed to it from the HDFS cluster which is hosting the input file. Moreover, there may be a tipping point in which the HDFS cluster is overburdened with too many out going data transfers, and the addition of an SI would result in a slowdown. We are pursuing discovering where this breaking point is, and what the ratio of HDFS VMs to SI VMs are for different applications.

We also ran experiments with accelerator nodes only running mappers. Our first notion was that mapper output which was consumed by a reducer would not be re-executed in case of a failure. This assumption was wrong. All mappers are re-executed on a machine regardless of whether it will be consumed again. Reducer output is already stored in HDFS with default replication of three. Check-pointing the map functions can be done by replicating the intermediate data as done in [8]. Other methods include saving the intermediate data to Amazon's S3 or EBS. The current Hadoop framework can also be modified to use tracking data on which mappers have been consumed to prevent re-execution during a fault.

Future work includes analyzing the cost-to-work ratio of different VM instance types. SIs can be used as probes for determining the best configuration. But this is only after fixing the availability of mapper output after termination, since we want to be able to restart the Task-Trackers with the optimal discovered configuration.

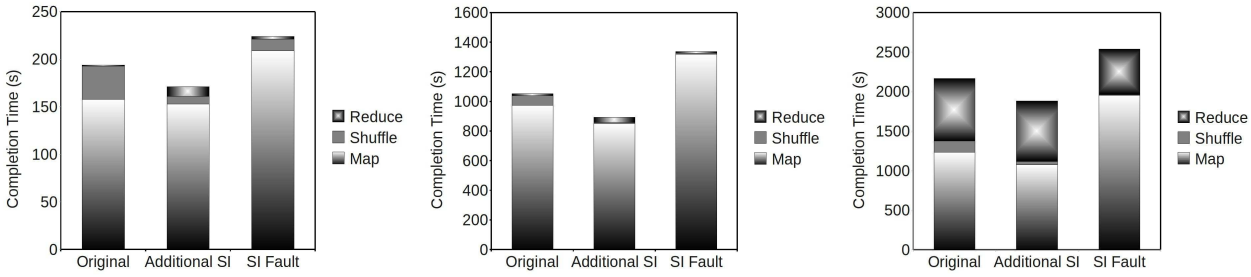


(a) Speedup per Additional SIs



(b) Cost of Speedup

Figure 3: The left graph shows the speedup of three different applications. The x-axis shows the number of SIs used in addition to the original HDFS cluster. Each data point represents the average of 5 trial runs. The right graph shows the cost versus the speedup. The baseline hourly cost for the HDFS cluster is $0.475 per hour. Each additional SI instance cost $0.040.



(a) Pi Estimator (b) Word Count (c) Sort

Figure 4: Completion time for three different applications showing runtime for MapReduce on the original on-demand HDFS cluster, with one SI, and with an SI termination halfway through completion (85, 450, 940 seconds for Pi Estimation, Word Count, and Sort, respectively).

The optimal configuration consists of having the most amount of mappers and reducers without them hitting performance bottlenecks due to sharing CPU, disk, network, and memory resources. Without the ability to save their intermediate data, the probes would become liabilities for wasted computation. Furthermore, we plan on investigating the use of heterogeneous configurations and instance types where a portion of the VMs only run reducers or mappers.

Additional future work includes analyzing the effects of staggering max bid prices across a set of SI VMs. In such a case it would be possible to only lose portions of accelerators at a time, essentially giving some VMs priority.

The nature of SI billing also leads to an interesting discussion on how to maximize utilization. An SI will be billed for the entire hour if terminated by the user even though it was only used for a partial hour, while no billing results if the VM ran for a partial hour and Amazon terminates the instance due to a rise in the current market price. Users may want to terminate an instance after an hours time in order to only pay for a full hour usage rather than pay for a partial hour, but this is only wise when the framework can recover from failure without significant adverse affects on the completion time.

6 Conclusion

We have presented SIs as a means of attaining performance gains at low monetary cost. We have characterized the EC2 SI pricing for informed decisions on making bids given the current market price. Our work has shown that due to the nature of spot instances and their reliability being a function of the bid price and market price, MapReduce jobs may suffer a slowdown if intermediate data is not stored in a fault tolerant manner. Moreover, a fault can cause a job's completion time to be longer than having not used additional SIs while potentially costing more.

References

[1] Amazon Web Services. http://aws.amazon.com.

[2] AWS Disscussion Forum. http://tinyurl.com/2dzp734.

[3] CHOHAN, N., BUNCH, C., PANG, S., KRINTZ, C., MOSTAFA, N., SOMAN, S., AND WOLSKI, R. AppScale: Scalable and Open AppEngine Application Development and Deployment. In *International Conference on Cloud Computing* (Oct. 2009).

[4] CloudExchange. http://cloudexchange.org/.

[5] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of 6th Symposium on Operating System Design and Implementation(OSDI)* (2004), 137–150.

[6] Google MapReduce Patent. http://tinyurl.com/yezbynq.

[7] Hadoop. http://hadoop.apache.org/.

[8] KO, S., HOQUE, I., CHO, B., AND GUPTA, I. On Availability of Intermediate Data in Cloud Computations. In *HotOS* (2009).

[9] OLSTON, C., REED, B., SRIVASTAVA, U., KUMAR, R., AND TOMKINS, A. Pig Latin: A Not-So-Foreign Language for Data Processing. In *ACM SIGMOD* (2008).

[10] THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P., AND MURTHY, R. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow. 2*, 2 (2009), 1626–1629.

[11] ZAHARIA, M., KONWINSKI, A., JOSEPH, A., KATZ, R., AND STOICA, I. Improving MapReduce Performance in Heterogeneous Environments. In *OSDI* (2008).