Statistics-Driven Workload Modeling for the Cloud

Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, David Patterson

Computer Science Division, University of California at Berkeley
{archanag, ychen2, fox, randy, pattrsn}@cs.berkeley.edu

***Abstract*— A recent trend for data-intensive computations is to use pay-as-you-go execution environments that scale transparently to the user. However, providers of such environments must tackle the challenge of configuring their system to provide maximal performance while minimizing the cost of resources used. In this paper, we use statistical models to predict resource requirements for Cloud computing applications. Such a prediction framework can guide system design and deployment decisions such as scale, scheduling, and capacity. In addition, we present initial design of a workload generator that can be used to evaluate alternative configurations without the overhead of reproducing a real workload. This paper focuses on statistical modeling and its application to data-intensive workloads.**

I. Introduction

The computing industry has recently uncovered the potential of large-scale data-intensive computing. Internet companies such as Google, Yahoo!, Amazon, and others rely on the ability to process large quantities of data to drive their core business. Traditional decision support databases no longer suffice because they do not provide adequate scaling of compute and storage resources. To satisfy their data-processing needs, many Internet services turn to frameworks like MapReduce [1], a big-data computation paradigm complementary to parallel databases. At the same time, the advent of cloud computing infrastructures such as Amazon EC2 and Rackspace, makes large-scale cluster computing accessible even to small companies [2]. The prevalence of SQL-like interfaces such as Hive [3] and Pig [4] further ease the migration of traditional database workloads to the Cloud.

Data intensive computing in the Cloud presents new challenges for system management and design. Key questions include how to optimize scheduling, reduce resource contention, and adapt to changing loads. The penalty for suboptimal decisions is amplified by the large number of simultaneous users and the sheer volume of data. As we will argue, heuristics and cost functions traditionally used for query optimization no longer suffice, and resource management strategies increasingly involve multiple metrics of success. Thus, there is the need for accurate performance prediction mechanisms to guide scheduling and resource management decisions, and realistic workload generators to evaluate the choice of policies prior to full production deployment.

In this work, we describe and evaluate a statistical framework that uses Kernel Canonical Correlation Analysis (KCCA) to predict the execution time of MapReduce jobs (Section III). This framework is an extension to our work in [5], where we have demonstrated the effectiveness of the KCCA technique for predicting query performance in parallel databases. Our key technical finding is that with the right choice of predictive features, KCCA leads to highly accurate predictions that improve with the quality and coverage of performance data. These features form the basis of a statistics-driven workload generator that synthesizes realistic workloads using the models developed in the KCCA framework (Section IV). This workload generator allows us to evaluate MapReduce optimizations on realistic workloads in the absence of a widely accepted performance benchmark. As we will detail, the workload generator relies on useful features identified by the prediction framework, as well as components that are common across different applications and computing paradigms. In other words, we argue that an effective prediction model is a prerequisite for a good workload generator. Conversely, a good workload generator allows the KCCA prediction framework to guide hardware, configuration and system management choices without committing a full implementation on a production cluster.

We begin our discussion with an overview of MapReduce (Section II), and we interleave reviews of relevant related work where appropriate. Our paper builds the case for statistics-driven distributed system modeling and design, an approach that is extensible to computing paradigms other than parallel databases and MapReduce.

II. MapReduce Overview

MapReduce was initially developed at Google for parallel processing of large datasets [1]. Today, MapReduce powers Google's flagship web search service, as well as clusters at Yahoo!, Facebook, and others [6]. Programs written using MapReduce are automatically executed in a parallel fashion on the cluster. Also, MapReduce can run on clusters of cheap commodity machines, an attractive alternative to expensive, specialized clusters. MapReduce is highly scalable, allowing petabytes of data to be processed on thousands and even millions of machines. Most importantly for our work, MapReduce is especially suitable for the KCCA prediction framework because it has a homogeneous execution model, and production MapReduce workloads often have repeated queries on similar or identical datasets.

At its core, MapReduce has two user-defined functions. The Map function takes in a key-value pair, and generates a set of intermediate key-value pairs. The Reduce function takes in all intermediate pairs associated with a particular key, and emits a final set of key-value pairs. Both the input pairs to Map and the output pairs of Reduce are placed in an underlying distributed file system (DFS). The run-time system takes care of retrieving from and outputting to the DFS, partitioning

the data, scheduling parallel execution, coordinating network communication, and handling machine failures.

A MapReduce execution occurs in several stages. There is a master daemon that coordinates a cluster of workers. The master divides the input data into many splits, each read and processed by a Map worker. The intermediate key-value pairs are periodically written to the local disk at the Map workers, usually separate machines from the master, and the locations of the pairs are sent to the master. The master forwards these locations to the Reduce workers, who read the intermediate pairs from Map workers using remote procedure call (RPC). After a Reduce worker has read all the intermediate pairs, it sorts the data by the intermediate key, applies the Reduce function, and appends the output pairs to a final output file for the Reduce partition. If any of the Map or Reduce executions lags behind, backup executions are launched. An entire MapReduce computation is called a *job*, and the execution of a Map or Reduce function on a worker is called a *task*. Each worker node allocates resources in the form of slots and each Map task or Reduce task uses one slot.

For our work, we select the Hadoop implementation of MapReduce [7]. The Hadoop distributed file system (HDFS) implements many features of the Google DFS [8]. The open source nature of Hadoop has made it a target for optimizations, e.g., an improved way to launch backup tasks [9], a fair scheduler for multi-user environments [10], pipeline task execution and streaming queries [11], and resource managers from multiple computational frameworks including MapReduce [12].

There have been several efforts to extend Hadoop to accommodate different data processing paradigms. Most relevant to our work, Hive [3] is an open source data warehouse infrastructure built on top of Hadoop. Users write SQL-style queries in a declarative language called HiveQL, which is compiled into MapReduce jobs and executed on Hadoop. Hive represents a natural place to begin our effort to extend the KCCA prediction framework to Hadoop.

III. Predicting Performance of Hadoop Jobs

Our goal is to predict Hadoop job performance by correlating pre-execution features and post-execution performance metrics. We take inspiration from the success of using statistical techniques to predict query performance in parallel databases [5]. KCCA allows us to simultaneously predict multiple performance metrics using a single model. This property captures interdependencies between multiple metrics, a significant advantage over more commonly used techniques such as Regression, which model a single metric at a time. For a more detailed comparison of KCCA to other statistical techniques, we refer the reader to [13].

Since Hive's query interface is similar to that of commercial parallel databases, it is a natural extension to evaluate the prediction accuracy of Hive queries in Hadoop using the KCCA technique as described in [5].

A. KCCA Prediction Framework

Figure 1 summarizes our adaptation of KCCA for Hadoop performance modeling. The first step in using KCCA-based

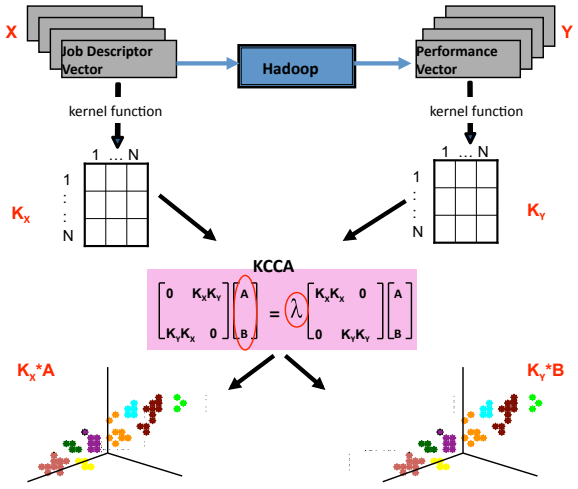

Fig. 1. Training: KCCA projects vector of Hadoop job features and performance features onto dimensions of maximal correlation across the data sets. Furthermore, its clustering effect causes "similar" jobs to be collocated.

modeling is to represent each Hadoop job as a feature vector of job characteristics and a corresponding vector of performance metrics. This step is the only place where we deviate from the winning methodology in [5]. We explain our choice of feature vectors in Section III-B.

The core idea behind using the KCCA algorithm is that multi-dimensional correlations are difficult to extract from the raw data of job features and performance features. KCCA allows us to project the raw data onto subspaces α and β such that the projections of the data are maximally correlated.

The precise mathematical construction is as follows. We start with N job feature vectors $\mathbf{x}_k$ and corresponding performance vectors $\mathbf{y}_k$. We form an $N \times N$ matrix K_x whose (i,j)th entry measures the similarity between $(\mathbf{x}_i, \mathbf{x}_j)$, and another $N \times N$ matrix K_y whose (i,j)th entry is the similarity between $(\mathbf{y}_i, \mathbf{y}_j)$. Our similarity metric is constructed from Gaussian kernel functions as discussed in [5]. Our prior work in [5] also discusses the impact of different kernel functions.

We then project K_x and K_y onto subspaces α and β. For this projection step, KCCA calculates the projection matrices A and B, respectively consisting of the basis vectors of subspaces α and β. In particular, the matrices A and B are calculated using the generalized eigenvector problem formulation in Figure 1 such that the projections $K_x \times A$ and $K_y \times B$ are maximally correlated. In other words, job feature projections $K_x \times A$ and performance feature projections $K_y \times B$ are collocated on subspaces α and β. Thus, we can leverage subspaces α and β for performance prediction.

Once we build the KCCA model, performance prediction is as follows. Beginning with a Hive query whose performance we want to predict, we create job feature vector $\hat{\mathbf{x}}$ and calculate its coordinates in subspace α. We infer the job's coordinates on the performance projection subspace β by using its 3 nearest neighbors in the job projection. This inference step is possible because KCCA projects the raw data onto dimensions

of maximal correlation, thereby collocating points on the job and performance projections. Finally, our performance prediction is calculated using a weighted average of the 3 nearest neighbors' raw performance metrics.

We evaluate this methodology using data from a production Hadoop deployment at a major web service. This deployment was on a multi-user environment comprising hundreds of homogeneous nodes, with a fixed number of map and reduce slots per node based on available memory.

We extracted our data from Hadoop job history logs, which are collected by the cluster's Hadoop master node to track details of every job's execution on the cluster. From these job logs, we construct performance feature vectors to include map time, reduce time, and total execution time. These metrics are central to any scheduling decisions. We also include data metrics such as map output bytes, HDFS bytes written, and locally written bytes.

There are several possible options for job feature vectors. The choice greatly affects prediction accuracy. Luckily, the best choice is intuitive and leads to good prediction accuracy.

B. Prediction Accuracy for Hive

The first choice of job feature vectors is an extension of feature vectors in [5], proved effective for parallel databases.

Like relational database queries, Hive queries are translated into execution plans involving sequences of operators. We observed 25 recurring Hive operators, including Create Table, Filter, Forward, Group By, Join, Move and Reduce Output, to name a few. Our initial job feature vector contained 25 features - corresponding to the number of occurrences of each operator in a job's execution plan.

Figure 2 compares the predicted and actual execution time using Hive operator instance counts as job features. The prediction accuracy was very low, with an negative R^2 value, indicating poor correlation between predicted and actual values [1]. Our results suggest that Hive operator occurrence counts are insufficient for modeling Hive query performance.

This finding is somewhat unsurprising. Unlike relational databases, Hive executions plans are an intermediate step before determining the number and configuration of maps and reduces to be executed as a Hadoop job. Job count and configuration are likely to form more effective job feature vectors, since they describes the job at the lowest level of abstraction visible prior to executing the job.

Thus, our next choice of job feature vector used Hive query's configuration parameters and input data characteristics. We included the number and location of maps and reduces required by all Hadoop jobs generated by each Hive query, and data characteristics such as bytes read locally, bytes read from HDFS, and bytes input to the map stage.

Figure 3 shows our prediction results for the same training and test set of Hive queries as in Figure 2. Our R^2 prediction accuracy is now 0.87 (R^2 = 1.00 signifies perfect prediction).

[1]Negative R^2 values are possible since the training data and test data are disjoint. Note that this metric is sensitive to outliers. In several cases, the R^2 value improved significantly by removing the top one or two outliers.

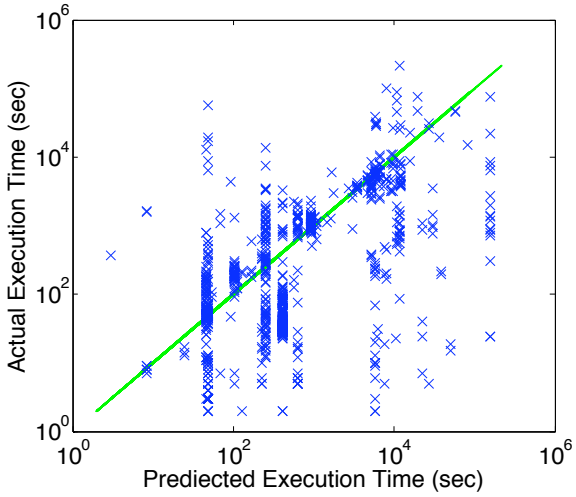

Fig. 2. Predicted vs. actual execution time for Hive queries, modeled using Hive operator instance counts as job features. The model training and test sets contained 5000 and 1000 Hive queries respectively. The diagonal green line represents the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in execution time.

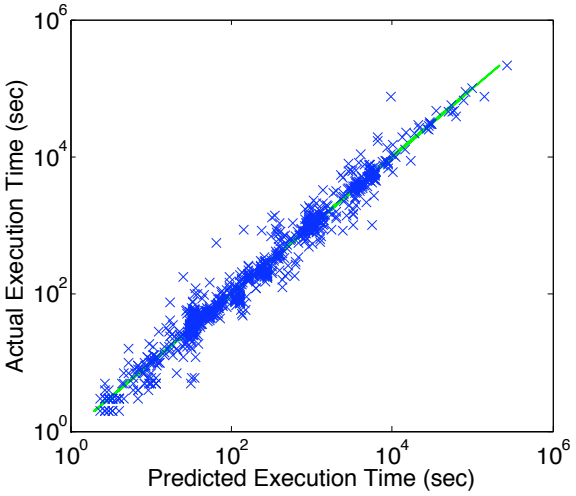

Fig. 3. Predicted vs. actual execution time for Hive queries, modeled using job configuration and input data characteristics as job features. The model training and test sets contained 5000 and 1000 Hive queries respectively. The diagonal green line represents the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in execution time.

Using the same model, our prediction accuracy was 0.84 for map time, 0.71 for reduce time, and 0.86 for bytes written. Our prediction accuracy was lower for reduce time since the reduce step is fundamentally exposed to more variability due to data skew and uneven map finishing times. These results convincingly demonstrate that feature vectors with job configuration and input data characteristics enable effective modeling of Hive query performance.

In future, we can improve prediction accuracy even further by using more customized kernel functions, two-step KCCA prediction, and better training set coverage [5].

C. Prediction for Other Hadoop Jobs

A significant advantage of our chosen job and performance feature vectors is that they contain no features that limit their scope to Hive queries. As a natural extension, we evaluate our performance prediction framework on another class of Hadoop jobs that mimic data warehouse Extract Transform Load (ETL) operations. ETL involves extracting data from outside sources, transforming it to fit operational needs, then loading it into the end target data warehouse. KCCA prediction is especially effective for Hadoop ETL jobs because the same jobs are often rerun periodically with varying quantities/granularities of data. Also, KCCA prediction can bring great value because ETL jobs are typically long-running. Thus, it is important to anticipate the job execution times so that system administrators can plan and schedule the rest of their workload.

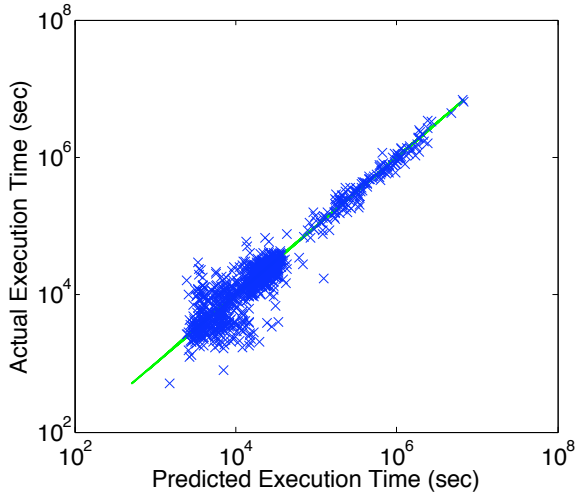

Fig. 4. Predicted vs. actual time for ETL jobs, modeled using job configuration and input data characteristics as job features. The model training and test sets contained 5000 and 1000 ETL jobs respectively. The diagonal green line represents the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in execution time.

Figure 4 shows the predicted vs. actual execution times of ETL jobs at the same Hadoop deployment. Our R^2 prediction accuracy for job execution time was 0.93. Prediction results for other metrics were equally good, with R^2 values of 0.93 for map time and 0.85 for reduce time. While the R^2 values are better than Hive predictions, there are some visible prediction errors for jobs with short execution times. These jobs are inherently more difficult to predict because the setup time for big data Hadoop jobs is also on the same time scale. Nevertheless, the results indicate that the KCCA framework is also effective for predicting Hadoop ETL jobs.

D. Potential Applications

Our finding has several implications beyond our immediate work. The high prediction accuracy suggests that related work on MapReduce optimization should consider augmenting their mechanisms using KCCA predictions.

For example, others researchers have adapted traditional query progress heuristics to estimate the execution time of Pig queries [14]. The technique estimates the remaining query execution time by comparing the number of tuples processed to the number of remaining tuples. This heuristic is effective when there is a constant tuple processing rate during query execution. In contrast, our statistical approach remains effective even if the tuple processing rate changes due to changing data locality or other effects. Another advantage of our technique is that we can make an accurate prediction prior to query execution, instead of halfway through the execution.

In addition, we have a good way to identify lagging tasks once we have accurate, statistics driven prediction of task finishing times. The LATE scheduler can use these predictions instead of the finishing time estimation heuristics in [9], leading to performance improvements.

Moreover, the Hadoop FAIR scheduler [10] can perform resource allocation using predicted execution time in addition to using the number of slots as a resource consumption proxy. This combination could lead to better resource sharing and decrease the need for killing or preempting tasks.

Accurate finishing time predictions also enable several other scheduler policies. One such example is a shortest-job-first scheduler, where we minimize per-job wait time by executing jobs in order of increasing predicted execution time. We can also implement a deadline-driven scheduler that allows jobs with approaching deadlines to jump the queue "just in time".

Furthermore, the prediction framework can help with resource provisioning. Given predicted resource requirements and desired finishing time, one can evaluate whether there are enough resources in the cluster or if more nodes should be pooled in. If predicted resource requirements are low, one can assign nodes for other computations, or turn off unnecessary nodes to reduce power without impacting performance.

E. Ongoing Work

To realize the full potential of our prediction framework, there are several concerns to address before integrating KCCA with scheduling and resource management infrastructures.

Different Hadoop workloads: Different organizations have different computational needs. We need to evaluate the prediction effectiveness for different workloads to increase confidence in the KCCA framework. In the absence of a representative MapReduce benchmark, we must rely on case studies on production logs. However, access to production logs involve data privacy and other logistical issues. Thus, we need a way to anonymize sensitive information while preserving a statistical description of the MapReduce jobs and performance.

Different job types: Here, we have obtained prediction results for Hive queries and ETL jobs. However, we believe the prediction framework is applicable for a more generic set of Hadoop jobs. The features we used for our predictions are common to all Hadoop jobs. Given that resource demands vary by the map and reduce functions, we can augment feature vectors with specific map and reduce function identifiers and/or the language in which these functions were implemented.

Different resources: A good scheduler should understand whether particular jobs have conflicting or orthogonal resource

demands. However, due to hardware and configuration differences, we cannot directly compare resource consumption between different clusters. If we can "rerun" the workload with good instrumentation, we can monitor the utilization of various resources including CPU, disk, network. This capability allows us to augment the job feature vector with resource consumption levels, providing more accurate predictions to the scheduler. Furthermore, we can "rerun" the workload on different hardware and cluster configurations, turning the KCCA framework into an even more powerful tool that can guide decisions on hardware and configuration choices.

The next section describes a workload generation framework that takes advantage of these opportunities.

IV. Towards a Workload Generation Framework

Recent research on MapReduce has relied on sort jobs and grid-mix [15] to generate evaluation workloads. These techniques are criticized for their inadequate diversity in data transfer patterns and insufficient evaluation of performance in a steady but non-continuous job streams. In contrast, our predecessor work benefited from the widely accepted TPC-DS database benchmark for its evaluations. In this section, we describe our initial work towards developing a MapReduce workload generation framework that captures the statistical properties of production MapReduce traces. The workload generator can act as a richer synthetic benchmark compared to current MapReduce evaluation methods.

We have several design goals. First, the framework must mask the actual computation done by jobs to prevent leakage of confidential competitive information. Often, MapReduce job source code reveals trade secrets about the scale and granularity of data processed for companies' core business decisions. Hiding job-specific characteristics would encourage companies to contribute traces towards the workload generation framework. Second, the framework has to be agnostic to the hardware, MapReduce implementation, and cluster configurations. This goal allows our framework to be used without the need to replicate production cluster environments. Third, the framework has to accommodate different cluster sizes and workload durations. This property allow us to investigate provisioning the right cluster size for a particular type of workload, and understanding both short and long-term performance characteristics.

The following design satisfies these goals. We start by extracting a statistical summary of the production MapReduce trace. This summary includes a distribution of inter-job arrival times and a distribution of job counts according to job name. For each job name, we also extract the distribution of job input sizes and input/shuffle/output data ratios. We scale the job input sizes by the number of nodes in the cluster. This adjustment preserves the amount of data processed by each node, facilitating a comparison of different node types, e.g. big nodes with more cores vs. small nodes with few cores. We then probabilistically sample the distributions to generate the workload as vectors of [launch times, job name, input size, and input/shuffle/output data ratios]. The job finishing time is not among the input data to the workload generator because it is the prediction output of our model.

It may require an unwieldy amount of data to accurately represent the statistical distributions. Therefore, we only extract the 1st, 25th, 50th, 75th, and 99th percentiles for the distributions of inter-job arrival times, input sizes, and data ratios. We do linear extrapolation between these percentiles to approximate the full distributions.

The precise algorithm is as follows:

1) From production trace
   - Compute the 1st, 25th, 50th, 75th, and 99th percentiles of inter-job arrival times.
   - Compute the CDF of job counts by $jobName$.
   - For each job, compute the 1st, 25th, 50th, 75th, and 99th percentiles of scaled input sizes, shuffle-input data ratio, and output-shuffle data ratio.
2) Perform probabilistic sampling on
   - The approximated distribution of inter-job arrival times to get t.
   - The truncated CDF of job counts to get $jobName$.
   - The approximated distribution of scaled input sizes for $jobName$ to get $inputSize$.
   - The approximated distribution of shuffle-input data ratio for $jobName$ to get $shuffleInputRatio$.
   - The approximated distribution of output-shuffle data ratio for $jobName$ to get $outputShuffleRatio$.
3) Add to workload [t, $jobName$, $inputSize$, $shuffleInputRatio$, $outputShuffleRatio$].

We repeat steps 2 and 3 until we have the required workload size in terms of either the number of jobs or the time duration of the workload.

This algorithm requires us to have a MapReduce job that adheres to specified shuffle-input and output-shuffle data ratios. This job is a straightforward modification of `randomwrite`, a code example included in recent Hadoop distributions.

There are several trade-offs associated with such a workload generator. Most visibly, we do not capture the compute part of MapReduce jobs. As explained by our goals, this trade-off is necessary, since knowledge of the computation could lead to a leakage of confidential information. On the other hand, it allows us to compare workloads from organizations using MapReduce for different computations. Moreover, we still capture the workload data transfer activity, which is often the biggest contributor to overall job finishing time. With a full implementation of our workload generator, we can concretely verify whether this is a sufficiently common case.

Another trade-off is the loss of hardware, data locality, and cluster configuration information from the initial job trace. The advantage of a hardware, locality, and configuration independent workload is that we can "re-run" workloads on different choices of hardware, MapReduce schedulers/implementations, and cluster configurations. Thus, we would be able to anticipate which choice would lead to the fastest finishing time for the workload at hand. This is a fundamentally different approach than detailed Hadoop simulators such as Mumak

[16], which is required for expediting the design and evaluation cycle of heuristics driven mechanisms.

We also do not capture any data skew that may affect MapReduce finishing time. This is also a conscious decision. Data skew would cause some map tasks or reduce tasks finish slower than the others and hold up the entire job. Thus, well-written MapReduce jobs should include mechanisms like hash functions to remove data skew, and our workload generator encompasses the desired operating mode.

Lastly, despite appearances, the choice of 1st, 25th, 50th, 75th, and 99th percentiles is anything but ad-hoc. These percentiles correspond to the five-number summary of statistical distributions, i.e. min, max, median, quantiles. The key strength of this summary is that it makes no assumptions about the underlying statistical distribution while capturing both the dispersion and skew in the data. Any sensitivity of quantile boundaries due to perturbations in the data would be bounded by the neighboring quantile boundaries. While capturing the complete distribution would be ideal, we believe the five-number summary is an acceptable trade-off.

We believe that such a workload generator would allow us to apply our KCCA prediction framework on different hardware, MapReduce implementations, and cluster configurations, while preserving the data transfer characteristics of the production traces. Thus, the KCCA prediction framework becomes a powerful tool that can guide choice of hardware, MapReduce schedulers/optimizers, and cluster configurations. At the same time, this architecture provides a framework for MapReduce operators to contribute anonymized production traces that would benefit the research community as a whole.

V. Summary and Open Questions

We have presented a statistics-driven modeling framework for data-intensive applications in the Cloud. We demonstrated good prediction accuracy on production Hadoop data analytic and warehousing jobs. We can leverage KCCA-based predictions for making decisions including job scheduling, resource allocation, and workload management. A precondition to implementing and validating frameworks for resource management is the presence of representative, realistic, portable workloads. We have also described a statistics-driven workload generator to meet this prerequisite.

There are several unanswered questions for resource management in the Cloud. First, it is unclear what granularity of scheduling decisions is appropriate in Hadoop. Concurrently scheduled jobs lead to variability in a job's performance, while concurrent tasks on the same node create variability in task finishing times. This problem is further complicated in clusters shared among various applications and not exclusively used by Hadoop [12]. One could devise per-application scheduling techniques, per-job techniques, per-task techniques, or per-node techniques; the limitations of each remain unexplored.

Next, several cloud computing infrastructures use virtual machines to provide isolation and abstract away resource availability. However, the additional layer of abstraction also creates complexity for performance modeling and decision making. An open challenge is to optimize resource usage through VM placement. The KCCA prediction framework could help address this challenge, provided we verify its effectiveness in VM environments.

Lastly, an opportunity yet to be capitalized in performance modeling is to appropriately account for variability inherent within a platform. It would be useful to normalize measured performance metrics with respect to each node's historical behavior, such as resource availability, average response time, and failure profile.

We have a working prototype of our statistics-driven workload generator. We plan to use this framework to evaluate potential solutions to the above issues, as well as to develop and evaluate our ideas on resource management. Our work demonstrates the advantage of statistics over heuristics in system optimization. We strongly believe that the statistical approach is relevant to a variety of modern distributed systems and parallel computing platforms.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, January 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[3] A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive - A Warehousing Solution Over a MapReduce Framework," in *Proc. International Conference on Very Large Data Bases*, 2009.

[4] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *Proc. ACM SIGMOD International Conference on Management of Data*, 2008.

[5] A. Ganapathi, H. Kuno, U. Daval, J. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting Multiple Performance Metrics for Queries: Better Decisions Enabled by Machine Learning," in *Proc International Conference on Data Engineering*, 2009.

[6] "Hadoop Power-By Page," http://wiki.apache.org/hadoop/PoweredBy.

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

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[9] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Symposium on Operating Systems Design and Implementation*, 2008.

[10] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job Scheduling for Multi-User MapReduce Clusters," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, 2009.

[11] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce Online," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-136, 2009.

[12] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, S. Shenker, and I. Stoica, "Nexus: A Common Substrate for Cluster Computing," Workshop on Hot Topics in Cloud Computing, 2009.

[13] A. Ganapathi, "Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning," Ph.D. dissertation, UC Berkeley, 2009.

[14] K. Morton, A. Friesen, M. Balazinska, and D. Grossman, "Estimating the Progress of MapReduce Pipelines," in *Proc. International Conference on Data Engineering*, 2010.

[15] "Gridmix," HADOOP-HOME/src/benchmarks/gridmix in all recent Hadoop distributions.

[16] "Mumak," http://issues.apache.org/jira/browse/MAPREDUCE-728, last retrieved Nov. 2009.