SLO-Driven Right-Sizing and Resource Provisioning of MapReduce Jobs[*]

Abhishek Verma
University of Illinois at Urbana-Champaign
Urbana, IL, US.
verma7@illinois.edu

Ludmila Cherkasova
Hewlett-Packard Labs
Palo Alto, CA, US.
lucy.cherkasova@hp.com

Roy H. Campbell
University of Illinois at Urbana-Champaign
Urbana, IL, US.
rhc@illinois.edu

ABSTRACT

There is an increasing number of MapReduce applications, e.g., personalized advertising, spam detection, real-time event log analysis, that require completion time guarantees or need to be completed within a given time window. Currently, there is a lack of performance models and workload analysis tools available to system administrators for automated performance management of such MapReduce jobs. In this work, we outline a novel framework for SLO-driven resource provisioning and sizing of MapReduce jobs. First, we propose an automated profiling tool that extracts a compact job profile from the past application run(s) or by executing it on a smaller data set. Then, by applying a linear regression technique, we derive *scaling factors* to accurately project the application performance when processing a larger dataset. The job profile (with scaling factors) forms the basis of a *MapReduce performance model* that computes the lower and upper bounds on the job completion time. Finally, we provide a fast and efficient capacity planning model that for a MapReduce job with timing requirements generates a set of resource provisioning options. We validate the accuracy of our models by executing a set of realistic applications with different timing requirements on the 66-node Hadoop cluster.

1. INTRODUCTION

Many companies are following the new trend of using MapReduce [1] and its open-source implementation Hadoop for large-scale, data intensive processing and for mining petabytes of data. Analyzing large amount of unstructured and semi-structured data is a high priority task for many businesses. The e-commerce companies examine web sites' traffic and users' navigation patterns to identify promising customers who might be likely "buyers". Banks and credit card companies analyze spending and purchasing patterns to prevent fraud and identity theft. There is a slew of interesting applications used for advanced data analytics in call centers and for information management associated with data retention, regulatory compliance, e-discovery and litigation issues. Many organizations rely on the ability to quickly process large quantities of new data to drive their core business. Often, the application is a part of a business pipeline (for example, see Facebook's workflow [9]), and the MapReduce job has to produce results by a certain time deadline, i.e., it has to achieve certain performance goals and service level objectives (SLOs). Currently, there is a lack of efficient performance models and workload analysis tools to ease performance management of MapReduce jobs. Businesses struggle on their own with capacity planning and resource sizing problems: they need to perform adequate application testing, empirical jobs' profiling and performance evaluation, and then use this experience to estimate appropriate resources required for timely processing of their applications. In this work, we outline a novel framework to solve this problem in a systematic way and to offer a resource sizing and provisioning service in MapReduce environments.

First, we propose an automated profiling tool that extracts a compact job profile from the past application execution(s) in the production Hadoop cluster. The proposed job profile aims to accurately reflect the application and system performance during all phases of a given job: map, shuffle/sort, and reduce phases. Alternatively, profiling can be done by executing a given application with a smaller input dataset than the original one. However, processing a larger input dataset (while keeping the number of reduce tasks unchanged) may lead to an increased amount of data shuffled and processed per reduce task, and it may alter the extracted application profile. A natural approach would be to normalize the proposed profile to reflect the "processing time per byte or unit of data" and then scale the application profile according to the amount of processed data. However, as we show in the paper, this approach does not work well for MapReduce jobs and can lead to a significant prediction error. Instead, by applying a linear regression technique, we derive *scaling factors* for shuffle and reduce phases to accurately estimate their service times as a function of processed data.

Designing an efficient and accurate performance model of a MapReduce job execution in the large-scale distributed environment, like Hadoop, is a challenging task. At a first glance, there are multiple factors and non-determinism in the execution that might impact the performance of Map-

[*]This work was largely completed during A. Verma's internship at HP Labs. R. Campbell and A. Verma are supported in part by NSF CCF grants #0964471, IIS #0841765 and Air Force Research grant FA8750-11-2-0084.

Reduce jobs. Does it mean that a simple analytic model is impossible? Should a detailed simulation that closely reflects the functionality and settings of the underlying environment be used for completion time estimates? In this work, we promote a *bounds-based MapReduce performance model*. Since the non-determinism in task scheduling inevitably impacts the job completion time, we argue for a simple and efficient model that computes the lower and upper bounds on job completion time instead of its accurate prediction with a detailed simulation model.

Finally, we propose a fast and efficient *capacity planning procedure* for estimating the required resources to meet a given application SLO. It operates over the following inputs *i)* a given job profile, *ii)* the amount of input data for processing, *iii)* the required job completion time. The output of the model is a set of plausible solutions (if such solutions exist for a given SLO) with a choice of different numbers of map and reduce slots that might be allocated for achieving performance goals of the application. We validate the accuracy of our approach and performance models by processing a set of realistic applications in the 66-node Hadoop testbed: the measured job completion times are within 10% of the given SLOs.

This paper is organized as follows. Section 2 introduces our approach towards profiling MapReduce jobs, scaling the job profile for processing an increased dataset, and presents a bounds-based MapReduce performance model. The efficiency of our approach and the accuracy of the designed model is evaluated in Section 3. We discuss how heterogeneity can be incorporated in our performance model in Section 4. Section 5 describes the related work. Section 6 summarizes the paper and outlines future directions.

2. BOUNDS-BASED MAPREDUCE PERFORMANCE MODEL

In this section, we outline our approaches for estimating the job completion time and solving the inverse problem of finding the appropriate amount of resources for a job that needs to meet a given (soft) deadline.

2.1 Useful Theoretical Bounds

First, we establish the performance bounds for a makespan (completion time) of a job represented as a set of n tasks processed by k servers (or by k slots in MapReduce environments). The assignment of tasks to slots is done using an online, *greedy* algorithm: assign each task to the slot which finished its running task the earliest. Let avg and max be the *average* and *maximum* duration of the n tasks respectively.

Makespan Theorem: The job makespan under the greedy task assignment is at least $n \cdot avg/k$ and at most $(n-1) \cdot avg/k + max$ [1].

The difference between lower and upper bounds represents the range of possible job completion times due to non-determinism and scheduling.

2.2 Completion Time Estimates of a MapReduce Job

As motivated by the Makespan Theorem, in order to approximate the overall completion time of a MapReduce job, we need to estimate the *average* and *maximum* task durations for different execution phases of the job, i.e., map, shuffle/sort, and reduce phases. These metrics and timing of different phases can be obtained from the counters at the job master during the job's execution or parsed from the logs.

Let us consider job J that is partitioned into N_M^J map tasks and N_R^J reduce tasks. Let J be already executed in a given Hadoop cluster with some arbitrary number of map/reduce slots. Below, we explain our profiling approach and introduce the MapReduce performance model for estimating the job completion time where J is executed with a different amount of resources. Let S_M^J and S_R^J be the number of map and reduce slots allocated to the **future** execution of job J respectively.

The **map stage** consists of a number of map tasks. If the number of tasks is greater than the number of slots, the task assignment proceeds in multiple rounds, which we call as *waves*. From the distribution of the map task durations of the past run, we compute the average duration M_{avg} and the maximum duration M_{max}. Then by applying the Makespan Theorem, the lower and upper bounds on the duration of the entire map stage in the future execution with S_M^J map slots (denoted as T_M^{low} and T_M^{up} respectively) are estimated as follows:

$$T_M^{low} = N_M^J \cdot M_{avg}/S_M^J \qquad (1)$$

$$T_M^{up} = (N_M^J - 1) \cdot M_{avg}/S_M^J + M_{max} \qquad (2)$$

The **reduce stage** consists of the *shuffle/sort* and *reduce* phases. The shuffle phase begins only after the first map task has completed. The shuffle phase completes when the entire map stage is complete and all the intermediate data generated by the map tasks has been shuffled to the reduce tasks and has been sorted.

The **shuffle phase** of the *first* reduce wave may be significantly different from the shuffle phase that belongs to the next reduce waves. This happens because the shuffle phase of the first reduce wave overlaps with the entire map stage, and hence its depends on the number of map waves and their durations. Therefore, from the past execution, we extract two sets of measurements: (Sh_{avg}^1, Sh_{max}^1) for shuffle phase of the first reduce wave (called, *first shuffle*) and $(Sh_{avg}^{typ}, Sh_{max}^{typ})$ for shuffle phase of the other waves (called, *typical shuffle*). Moreover, we characterize a first shuffle in a special way and include only the non-overlapping portion (with map stage) in our metrics: Sh_{avg}^1 and Sh_{max}^1. This way, we carefully estimate the latency portion that contributes explicitly to the job completion time. The *typical shuffle* phase is computed as follows:

$$T_{Sh}^{low} = \Big(N_R^J/S_R^J - 1\Big) \cdot Sh_{avg}^{typ} \qquad (3)$$

$$T_{Sh}^{up} = \Big((N_R^J - 1)/S_R^J - 1\Big) \cdot Sh_{avg}^{typ} + Sh_{max}^{typ} \qquad (4)$$

The **reduce phase** begins only after the shuffle phase is complete. From the distribution of the reduce task durations of the past run, we compute the *average* and *maximum* metrics: R_{avg} and R_{max}. Similarly, Makespan Theorem can be directly applied to compute the lower and upper bounds of completion times of the reduce phase (T_R^{low}, T_R^{up}) when a different number of allocated reduce slots S_R^J is given.

Finally, we can put together the formulae for the lower

[1]Similar ideas were explored in the classic papers on scheduling, e.g., to characterize makespan bounds in [3]. For a formal proof, see [11].

and upper bounds of job completion time:

$$T_J^{low} = T_M^{low} + Sh_{avg}^1 + T_{Sh}^{low} + T_R^{low} \qquad (5)$$

$$T_J^{up} = T_M^{up} + Sh_{max}^1 + T_{Sh}^{up} + T_R^{up} \qquad (6)$$

Note that we can re-write Eq. 5 for T_J^{low} by replacing its components with more detailed Eq. 1, Eq. 3 and similar equations for reduce phase as shown below:

$$T_J^{low} = \frac{N_M^J \cdot M_{avg}}{S_M^J} + \frac{N_R^J \cdot (Sh_{avg}^{typ} + R_{avg})}{S_R^J} + Sh_{avg}^1 - Sh_{avg}^{typ} \qquad (7)$$

This allows us to express the estimates for completion time as a function of map/reduce tasks (N_M^J, N_R^J) and the allocated map/reduce slots (S_M^J, S_R^J):

$$T_J^{low} = A_J^{low} \cdot \frac{N_M^J}{S_M^J} + B_J^{low} \cdot \frac{N_R^J}{S_R^J} + C_J^{low} \qquad (8)$$

where $A_J^{low} = M_{avg}$, $B_J^{low} = (Sh_{avg}^{typ} + R_{avg})$, and $C_J^{low} = Sh_{avg}^1 - Sh_{avg}^{typ}$.

The equation for T_J^{up} can be written similarly using Equations 4 and 6.

2.3 Scaling Factors

In the previous section, we show how the job profile can be extracted from the past run of the application on the entire dataset. Alternatively, profiling can be done by executing a given application with a smaller input dataset than the original one. However, processing a larger input dataset (while keeping the number of reduce tasks unchanged) may lead to the increased durations of the reduce tasks as the size of the intermediate data processed by each reduce task increases. The duration of the map tasks is not impacted because this larger dataset is split into a larger number of map tasks but each map task processes a similar portion of data.

Note, that the shuffle phase duration mainly depends on the network performance. The reduce phase duration depends on the user supplied reduce function and the disk write performance. Consequently, we derive two scaling factors for shuffle and reduce phases separately, each one as a function of the processed dataset size.

We extend the job profile by extracting an additional metric: $Selectivity_M$ that defines the ratio of the map output size to the map input size. It is used to estimate the amount of intermediate data produced by the map stage as input to the reduce stage.

We perform a few experiments $(i = 1, 2, ..., k)$ with a given MapReduce job for processing different size input datasets (while keeping the number of reduce tasks constant), and collect the job profile measurements. Let D_i be the amount of intermediate data processed per reduce task, and let $Sh_{i,avg}^{typ}$ and $R_{i,avg}$ be the job profile measurements for shuffle and reduce phases respectively. We apply a linear regression to solve the following set of equations:

$$C_0^{Sh} + C_1^{Sh} \cdot D_i = Sh_{i,avg}^{typ}, (i = 1, 2, \cdots, k)$$
$$C_0^R + C_1^R \cdot D_i = R_{i,avg}, (i = 1, 2, \cdots, k)$$

Derived scaling factors (C_0^{Sh}, C_1^{Sh}) for shuffle phase and (C_0^R, C_1^R) for reduce phase are incorporated in the job profile. When job J processes an input dataset that leads to a different amount of intermediate data D_{new} per reduce task, its profile is updated as $Sh_{avg}^{typ} = C_0^{Sh} + C_1^{Sh} \cdot D_{new}$ and $R_{avg} = C_0^R + C_1^R \cdot D_{new}$. Similar scaling factors can be derived for maximum durations Sh_{max}^{typ} and R_{max} as well as for the first shuffle.

2.4 SLO-based Resource Provisioning

In this section, we address the inverse problem: given a MapReduce job J with input dataset D, how many map and reduce slots (S_M^J, S_R^J) should be allocated to this job so that it finishes within time T. We outline an efficient and fast algorithm that provides a compact answer to this question.

Note that there are different choices for navigating the resource provisioning algorithm: the SLO-time T is targeted as *i)* the *lower bound* of the job completion time, *ii)* the *upper bound* of the job completion time, or *iii)* some combination of lower and upper bounds.

Let T be targeted as the *lower bound* of the job completion time. Let input dataset D be partitioned for processing in N_M^J map tasks and N_R^J reduce tasks. First, the algorithm sets S_M^J to the largest possible map slots allocation in the cluster (by considering the number of map tasks for processing N_M^J and the maximum number of map slots S_M in the cluster). By using computed allocation S_M^J, we can derive the required allocation of reduce slots S_R^J from Eq. 8. If a found value of S_R^J is positive and less than the overall number of reduce slots available in the system, then the pair (S_M^J, S_R^J) represents a feasible solution for achieving a given SLO. If a found value of S_R^J is negative or higher than S_R then it means that job J can not be completed within T with allocated map slots S_M^J.

If a feasible solution (S_M^J, S_R^J) is found, the algorithm performs the next iteration by decreasing the number of map slots by 1. This way, the algorithm sweeps through the entire range of map slot allocations and finds the corresponding values of reduce slots for completing J within time T. The algorithm is **linear** in the number of map slots $(O(min(N_M^J, S_m))$.

The case when T is targeted as the upper bound (or the bounds combination) is handled similarly.

Note the following *monotonicity property* for MapReduce environments: by allocating a greater number of map/reduce slots than the computed number (S_M^J, S_R^J), one can only decrease the job completion time.

3. EVALUATION

We perform our experiments on 66 HP DL145 GL3 machines. Each machine has four AMD 2.39GHz cores, 8 GB RAM and two 160GB hard disks (7200rpm SATA). The machines are set up in two racks and interconnected with Gigabit Ethernet. We use Hadoop 0.20.2 with two machines as the job master and the DFS master. The remaining 64 machines are used as worker nodes, each configured with a single map and reduce slot (since data disk is a bottleneck). The blocksize of the file system is set to 64MB and the replication level is set to 3. We disabled speculation as it did not lead to any significant improvements. To validate our models, we use four representative MapReduce applications:

1. **Twitter:** This application uses the 25GB twitter dataset created by Kwak et. al. [5] containing an edge-list of twitter userids. Each edge (i, j) means that user i follows user j. The Twitter application counts the number of asymmetric links in the dataset, that is, $(i, j) \in E$, but $(j, i) \notin E$.

2. **Sort:** The Sort application sorts 64 GB of data generated using random text writer in GridMix2[2]. It uses identity map and reduce tasks, since the framework performs the actual sorting.

3. **WikiTrends:** We use the data from Trending Topics[3] with Wikipedia article traffic logs collected and compressed every hour. Our MapReduce application computes a visitor count for each article.

4. **WordCount:** It counts the word frequencies in 27 GB of Wikipedia article text corpus. The map task tokenizes each line into words, while the reduce task counts the occurrence of each word.

In our *first set of experiments*, we estimate the job completion times when the job is executed with different numbers of map and reduce slots. Initially, we build a job profile from the job execution with 64 map and 32 reduce slots. Using the extracted job profile and applying formulae described in Section 2, we predict each job completion time for the job execution with 32 map and 16 reduce slots across 5 trials. The results are summarized in Fig. 1.

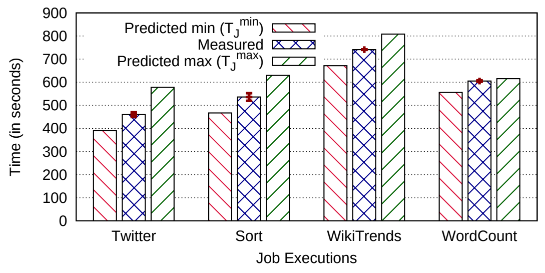


Figure 1: Predicted vs measured completion times for four applications (using historic profiling of the past runs of same job).

We observe that the measured completion time falls in between the computed lower and upper bounds. The completion time predicted as the average of low and upper bounds might be the closest estimate for the measured job completion time.

In our *second set of experiments*, we execute our applications on gradually increasing datasets with a fixed number of reduce tasks for each job. Our intent is to measure the trend of the shuffle and reduce phase durations (average and maximum) and validate the linear regression approach proposed in Section 2.3 for deriving the scaling factors. Fig. 2 shows that the trends for WordCount are indeed linear [4]. While we show multiple points in Fig. 2, typically, at least two points are needed for deriving the scaling factors. The accuracy improves with accumulated execution points. A 10% sample of the original dataset used for profiling leads to accurate results.

Alternatively, we tried a simple approach that scales the job profile according to the amount of processed data. However, it did not work well as the measured trends shown in Fig. 2 are not directly proportional to the processed dataset size.

[2]http://hadoop.apache.org/mapreduce/docs/current/gridmix.html
[3]http://trendingtopics.org
[4]The results for WikiTrends, Twitter and Sort are similar.

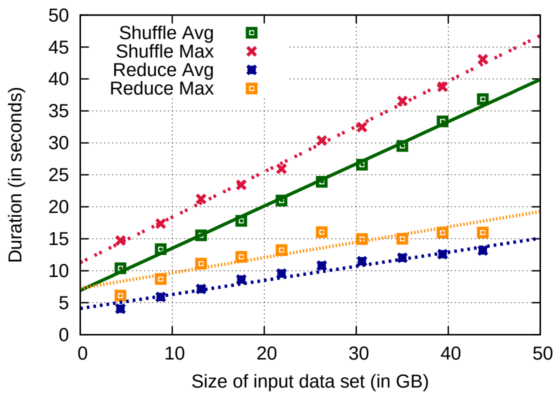


Figure 2: WordCount: scaling factors of shuffle and reduce durations.

Now, we evaluate the approach, where the job profile is built using small size input datasets, and then this job profile is used for predicting the completion time of the same application processing a larger input dataset. Therefore, in these experiments, first, the job profiles are built using the three trials on small datasets (e.g., 4.3, 8.7 and 13.1 GB for WordCount) with different numbers of map and reduce slots.

After that, by applying linear regression to the extracted job profiles from these runs, we determine the scaling factors for shuffle and reduce phases of our MapReduce jobs.

These scaling factors are used for extrapolating the shuffle and reduce phase durations when the same applications are processing larger input datasets with the following parameters: By using the updated (scaled) job profiles and

Parameters	Twitter	Sort	WT	WC
# of map tasks	370	1024	168	425
# of reduce tasks	64	64	64	64
# of map slots	64	64	64	64
# of reduce slots	16	32	8	8

Table 1: Application Parameters for Twitter, Sort, WikiTrends (WT) and WordCount (WC)

applying the formulae described in Section 2, we predict the job completion times for processing these larger datasets. Fig. 3 shows the results of these experiments. The red error

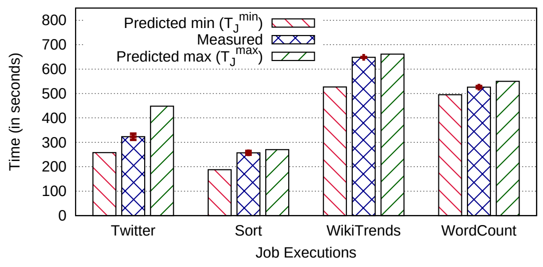
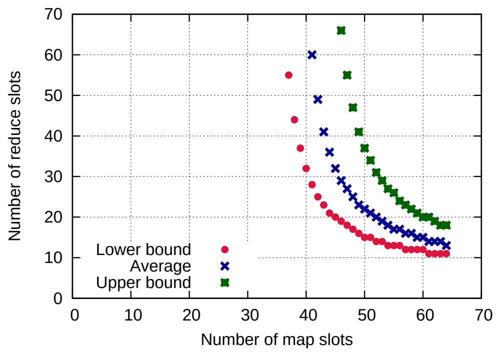
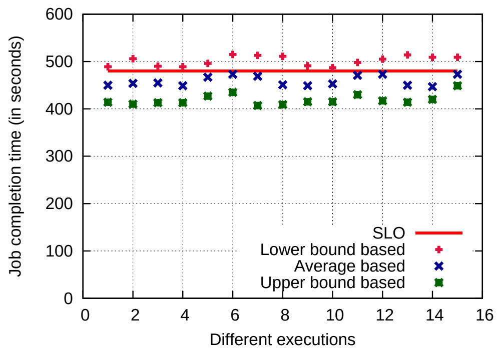


Figure 3: Predicted vs measured completion times for four applications (applying scaling factors to profiling executions on smaller data sets).



(a) Resource allocation curves.



(b) Can we meet the deadline with projected resource allocations?

Figure 4: SLO-based resource provisioning for WordCount.

bars show the variation in measured times across 5 trials.

We observe that our model accurately bounds the measured job completion time. If we use the average of lower and upper bounds (denoted T_J^{avg}) for prediction, then the relative error between T_J^{avg} and the measured job completion time is less than **10%** in all cases.

Finally, we perform experiments to validate the accuracy of the SLO-based resource provisioning model introduced in Section 2.4. We aim to evaluate the accuracy of resource allocations recommended by the model for completing the job within a given deadline.

Fig. 4 (a) shows a variety of plausible solutions (the outcome of the SLO-based model) for WordCount with a given deadline D= 8 minutes. The X and Y axes of the graph show the number of map and reduce slots respectively that need to be allocated in order to meet the job's deadline. There are three curves that correspond to the computation when given time T is targeted as the lower, upper, or the average of the lower and upper bounds. As expected, the recommendation based on the upper bound (worst case scenario) suggests more aggressive resource allocations with a higher number of map and reduce slots as compared to the resource allocation based on the lower bound. The difference in resource allocation is influenced by the "gap" between the lower and upper bounds.

Next, we sample each curve in Fig. 4 (a), and execute WordCount with recommended allocations of map and reduce slots in our 66-node Hadoop cluster to measure the actual job completion times. Fig. 4 (b) summarizes the results of these experiments. The model based on lower bounds suggests insufficient resource allocations: almost all the job executions with these allocations have missed their deadline. The closest results are obtained if we use the model that is based on the average of lower and upper bounds of completion time. If we base our computation on the upper bounds of completion time, the model over provisions resources. While all the job executions meet their deadline, the measured job completion times are lower than the target SLO, often by as much as 20%. The resource allocation choice will depend on the user goals and his requirements on how close to a given SLO the job completion time should be. The user considerations might also take into account the service provider charging schema to evaluate the resource allocation alternatives on the curves shown in Fig. 4 (a).

4. DISCUSSION

Originally, Hadoop was designed for homogeneous environment. There has been recent interest [12] in heterogeneous MapReduce environments. While in this work, we present our results using a homogeneous Hadoop cluster, our approach will work in heterogeneous MapReduce environments. In a heterogeneous cluster, the slower nodes will be reflected by the longer tasks durations, and they all would contribute to the average and maximum task durations in the job profile. Although we do not explicitly consider different types of nodes, their performance is reflected in the job profile and can be used in the future prediction. When the percentage of heterogeneous nodes is the same in the experiments for building a job profile with a smaller dataset and later for running the job with a larger dataset then the results would be accurate. So, while we do not explicitly target the heterogeneous environment, our approach should efficiently work in heterogeneous Hadoop clusters as well.

In this paper, we presented profiling and experimental results for jobs run in the Hadoop cluster in isolation. However, in our earlier work [11], we applied similar job profiling ideas while designing the SLO-based scheduler for processing multiple (concurrent) jobs with user-specified deadlines. While in some cases we did observe a higher prediction error (up to 20%), most of the time, the predictions for job completion time were within 10% of the measured ones in the testbed on average.

The accuracy of the results depends on the resource contention, especially, the network contention in production Hadoop clusters. In our testbed, the network was not a bottleneck, and it led to the accurate prediction results for job completion time. Typically, service providers tend to over provision network resources to avoid undesirable side effects of network contention. However, for very large Hadoop clusters it is a challenge. It is an interesting modeling question whether a network contention factor can be introduced, measured, and incorporated in the proposed performance models.

5. RELATED WORK

There are several research efforts to design job progress estimators for predicting the job completion time.

Polo et al. [8] introduce an online job completion time estimator which can be used for adjusting the resource allocations of different jobs. However, their estimator tracks the progress of the map stage alone and has no information or control over the reduce stage.

Ganapathi et al. [2] use Kernel Canonical Correlation Analysis to predict the performance of MapReduce workloads. However, they concentrate on Hive queries and do not attempt to model the actual execution of the MapReduce job. The authors discover the feature vectors to characterize Hive queries through statistical correlation.

Morton et al. [6] propose *ParaTimer* for estimating the progress of parallel queries expressed as Pig scripts [7] that can translate into directed acyclic graphs (DAGs) of MapReduce jobs. However, instead of a detailed profiling technique that is designed in our work, the authors rely on earlier debug runs of the same query for estimating throughput of map and reduce stages on the input data samples provided by the user. The approach is based on precomputing the expected schedule of all the tasks. Usage of the FIFO scheduler limits the approach applicability for progress estimation of multiple jobs running in the cluster with a different Hadoop scheduler.

Kambatla et al [4] have focused on optimizing the Hadoop configuration parameters (number of map and reduce slots per node) to improve MapReduce program performance. A signature-based (fingerprint-based) approach is used to predict the performance of a new MapReduce program using a set of already studied programs.

Tian and Chen [10] propose an interesting approach to predict MapReduce program performance from the test runs with a smaller number of nodes. By the problem definition it is the closest work to ours. However, the proposed approach is significantly different from our solution. The authors consider a very fine granularity of MapReduce processing. For example, the map task is partitioned in 4 functions: read a block, map function processing of the block, partition and sort of the processed data, and the combiner step (if it is used). The reduce task is decomposed in 4 functions as well. Then using a linear regression technique, the cost (duration) of each function is approximated. These functions are used for predicting the larger dataset processing. There are a few simplified assumptions introduced by the authors, e.g., a single wave in the reduce stage. In our approach, we derive the job profile from the job processing logs that are readily available at any Hadoop system. This provides reliable measurements of task durations, and enables a simple and intuitive performance model. Our solution for finding resource allocations that support the job completion goals is also different from the approach proposed in [10]. They formulate it as an optimization problem that can be solved with existing commercial solvers. Our approach does not require any additional external tools.

In our earlier work [11], we proposed a framework, called ARIA [11], for a Hadoop deadline-based scheduler. This scheduler extracts and utilizes the job profiles from the past executions. The shortcoming of the earlier work is that it does not have scaling factors to adjust the extracted profile and lacks the ability for job profiling on smaller datasets. In the current work, we propose a general profiling approach and a more advanced resource provisioning model with a variety of different options that offers to service providers a set of interesting trade-offs.

6. CONCLUSION

In this work, we outlined a novel framework with performance models and job profiling tools that aim to enable the automated workload management of MapReduce jobs with timing requirements. These models can also be used for more traditional capacity planning and resource provisioning tasks. The proposed approach is simple, intuitive and efficient, it aims to assist the system administrators in their performance evaluation and MapReduce cluster management efforts.

The proposed performance models were designed for the case without node failures. The next step is to extend the approach for incorporating different failure scenarios and estimating their impact on the application performance and achievable "degraded" SLOs. Another interesting future work is the resource provisioning and meeting SLO requirements of more complex applications (e.g., Pig queries) that are defined as a composition of MapReduce jobs.

7. REFERENCES

[1] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. In *Proceedings of SMDB*, 2010.

[3] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. Journal*, 45, 1966.

[4] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In *Proc. of the First Workshop on Hot Topics in Cloud Computing*, 2009.

[5] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Proc. of Intl. Conference on World Wide Web.* ACM, 2010.

[6] K. Morton, M. Balazinska, and D. Grossman. ParaTimer: a progress indicator for MapReduce DAGs. In *Proc. of SIGMOD.* ACM, 2010.

[7] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proc. of SIGMOD.* ACM, 2008.

[8] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, and I. Whalley. Performance-driven task co-scheduling for MapReduce environments. In *12th IEEE/IFIP NOMS*, 2010.

[9] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at Facebook. In *Proc. of SIGMOD*, pages 1013–1020. ACM, 2010.

[10] F. Tian and K. Chen. Towards Optimal Resource Provisioning for Running Ma pReduce Programs in Public Clouds. In *Proc. of IEEE Conference on Cloud Computing (CLOUD 2011)*, 2011.

[11] A. Verma, L. Cherkasova, and R. Campbell. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. In *Proc. of 8th IEEE/ACM Intl. Conference on Autonomic Computing (ICAC)*, June, 2011.

[12] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, 2008.