Predicting Execution Bottlenecks in Map-Reduce Clusters

Edward Bortnikov
Yahoo! Labs
Haifa, Israel
ebortnik@yahoo-inc.com

Ari Frank*
Affectivon Inc
Kiryat Tivon, Israel
ari@affectivon.com

Eshcar Hillel
Yahoo! Labs
Haifa, Israel
eshcar@yahoo-inc.com

Sriram Rao
Yahoo! Labs
Santa Clara, US
sriramr@yahoo-inc.com

Abstract

Extremely slow, or straggler, tasks are a major performance bottleneck in map-reduce systems. Hadoop infrastructure makes an effort to both avoid them (through minimizing remote data accesses) and handle them in the runtime (through speculative execution). However, the mechanisms in place neither guarantee the avoidance of performance hotspots in task scheduling, nor provide any easy way to tune the timely detection of stragglers. We suggest a machine-learning approach to address these problems, and introduce a *slowdown predictor* – an oracle to forecast how much slower a task will run on a given node, compared to similar tasks. Slowdown predictors can be embedded in the map-reduce infrastructure to improve the agility and timeliness of scheduling decisions. We provide initial evaluation to demonstrate the viability of our approach, and discuss the use cases for the new paradigm.

1 Introduction

In the recent years, map-reduce (MR) platforms have emerged as the most popular infrastructure for Web-scale data analysis. Pioneered by Google's proprietary design [2] and followed by the open-source Apache Hadoop platform [6], MR systems serve the analysis of huge datasets for Web search, computational advertising, content optimization, etc.

The map-reduce paradigm offers a programmer a simple API for data management and computation, while encapsulating the complexity of scalable implementation thereof. The underlying infrastructure harnesses multiple machines, or nodes, for reliable storage and workload execution. In this context, all data is replicated over multiple nodes, and every data-processing program, or *job*, can run on multiple nodes. Modern MR clusters scale to many thousands of machines.

A single job is comprised of multiple *tasks* shaped in two phases *map* and *reduce*. A map task, or *mapper*, processes a single and logically sequential data chunk stored on the filesystem. It produces a set of key-value pairs, which are further distributed by key to the reducers, in an all-to-all communication pattern called *shuffle*. A reduce task, or *reducer*, processes the received data, key by key, and creates a permanent output that is written to the filesystem.

A map-reduce system can simultaneously run multiple jobs competing for the node's resources and traffic bandwidth. These conflicts cause slowdown in the execution of tasks. The duration of each phase, and hence the duration of the job is determined by the slowest, or *straggler*, task. We use the *slowdown* metric – the ratio between the task's execution time and the median running time of a sibling task in the same job – to characterize stragglers. Figure 1 depicts the straggler mappers and reducers (slowdown above 5) observed in a production Hadoop cluster at Yahoo! during a month of operation. It shows that there are jobs in which the slowdowns are huge, up to many dozens. This phenomenon is most emphasized in large-scale production jobs. For example, among the jobs with more than 1000 mappers, 5% have stragglers, whereas among those with more than 1000 reducers, as many as 50% do.

The slowdowns of individual tasks are highly correlated with overall job latencies. The correlation is not direct, since mappers and reducers are typically executed in multiple waves. However, significant task slowdowns tend to indicate bottlenecks in job execution as well. Hence, reducing the straggler effect is paramount for improving the system's responsiveness.

The MR framework addresses straggler tasks in two ways: *avoidance* and *detection*. Straggler avoidance is part of task scheduling. The scheduler reduces network bottlenecks, by assigning the mapper task either to a node that stores the data replicas (*local* mappers), or at least to a node sharing the same physical rack with the

*Research was done while the author was at Yahoo! Labs.

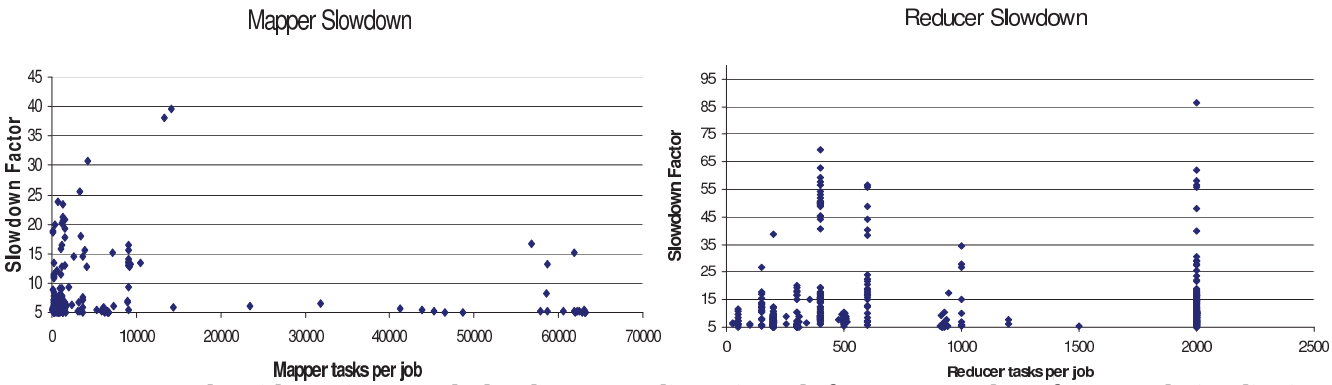

Figure 1: **Jobs with maximum task slowdown exceeding 5 (sample from a 1-month performance log) – distribution by the number of mappers and reducers. Each data point stands for a single job instance.**

data (*rack-local* mappers) (e.g., [6]). Straggler detection happens as the running task monitoring. Upon finding a task that runs exceedingly long, the infrastructure speculatively launches a *duplicate* task that does the same work, and uses the output of the task that finishes first.

However, these existing mechanisms fail to handle the straggler problem effectively. Specifically, the locality optimization is counterproductive if mappers are placed on slow or computationally congested nodes merely to preserve the proximity to data. Figure 2 illustrates this phenomenon, based on a 4-month production dataset. Each data point depicts a *bottleneck* instance – one of the 15%-topmost nodes by the number of straggling local mappers that occurred in a six-hour window. The vertical bands in the plot demonstrate that bottlenecks persist – i.e., the tasks run consistently slower on the same set of nodes, despite the data locality.

As stragglers emerge, the MR infrastructure often fails to detect them and launch duplicate tasks on time. In Hadoop, for instance, no speculative executions are considered as long as there is a backlog of non-scheduled tasks for the job. Furthermore, the detection mechanism must be fine-tuned through configuration parameters that are hard to adapt to particular workloads. Our measurements on production data show that as many as 90% of speculative tasks are useless – they are launched too late to finish before the original stragglers, and end up being killed by the system.

Our contribution We address the above shortcomings by introducing *slowdown predictor* – a novel machine-learned oracle component that detects potential or existing bottlenecks in MR clusters based on patterns mined from historical performance data. The oracle exposes a simple API: given a task-node pair $\langle t, n \rangle$, it produces an estimate for the slowdown of t running on n. In this context, both the task and the node are modeled as feature vectors, assembled from MR-level and system metrics.

Slowdown prediction can serve both the scheduling and the speculative execution scenarios. Consider, for example, augmenting Hadoop's capacity scheduler [6] with a predictor oracle. The scheduler manages a pool of execution *slots* at each node, and assigns the free slots to tasks waiting for execution. In its current form, it will always use a slot, should a task be waiting for it. With a predictor's help, the scheduler can avoid creating task-to-node assignments for which the slowdown estimate is high. E.g., it can prevent allocating congested hardware to resource-savvy tasks (despite the existence of free slots), hence avoiding the emergence of stragglers.

The predictor abstraction generalizes the currently existing heuristics, while being far better amenable to tuning for specific workloads. For example, a mapper's locality with respect to data might be just one feature that affects the slowdown, along with the task's input size, CPU load, RAM usage, network traffic, etc.

In this paper, we demonstrate a specific predictor implementation, which is trained on a Hadoop performance dataset collected at Yahoo! [4]. Our evaluation shows that slowdown can be effectively predicted, especially for mappers, thanks to the dominance of large periodic jobs in production environments.

The rest of this paper is structured as follows. Section 2 defines the slowdown prediction problem. The features used for training a slowdown predictor for Hadoop appear in Section 3. The machine learning technique employed by our implementation is described in Section 4. Section 5 presents the initial evaluation results of our model. Finally, we discuss related work in Section 6, and conclude in Section 7.

2 Problem Definition

A task *slowdown* is defined as the ratio between the running time of the task on the node it is assigned to and the median running time of similar tasks over all historical executions. Intuitively, two similar tasks are identified by the same "signature". As an approximation, sibling mappers or reducers in the same job with roughly the

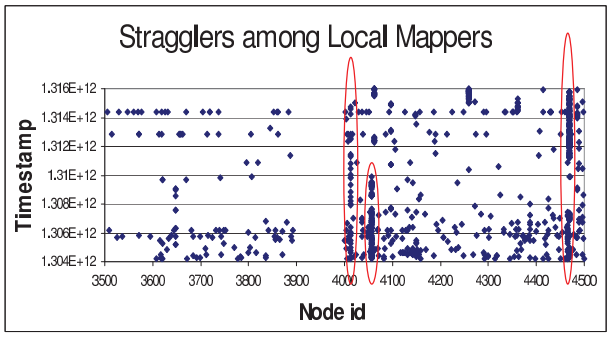

Figure 2: **Local mapper execution bottlenecks (4-month performance log). Each data point stands for one of the 15%-topmost nodes by the number of straggling local mappers (slowdown above 5) that occurred in a six-hour time frame. Note the bottleneck stickiness – i.e., some nodes consistently generate stragglers among local mappers.**

same input and output size are considered similar. More precisely, each task has a characterizing *profile*; it can be considered as a vector of attributes. Two tasks are *similar* if their profiles are similar.

The environment in which the task is running is defined by the node it is assigned to, and also by the rack and cluster to which the node belongs. In the context of a node-task pair, these are called the *hardware* running the task. Like a task, the hardware is characterized by a profile. The *slowdown estimate* is a mapping from the task and hardware profiles to the slowdown value of the task. A high estimate indicates that the task might become a bottleneck in the job's execution.

The profiles of tasks and hardware are dynamic, and may change over time. As the task runs, the more information becomes available about it, e.g., the output size can be projected once a fraction of input is processed. All the more so, during the execution attributes of the hardware change, and with it its profile. Therefore, the slowdown estimate during the execution of a task on a node may differ from an earlier one from when the node was only a candidate for hosting the task.

3 Slowdown Predictor Model

The *slowdown predictor* model is a machine-learned oracle for map-reduce systems forecasting execution bottlenecks. Based on the profiles of the task and the hardware, the predictor estimates the task's slowdown. The oracle can be applied either upon the task's assignment, or during the execution. Hence, the task and hardware profiles are dynamic, and the predictor must be capable of receiving wildcards instead of some feature values.

In what follows, we describe the features used for a model designed for the Hadoop infrastructure atop the capacity scheduler [6]. Part of these features are Hadoop counters, whereas the others are external system metrics.

3.1 Task Features

Job-Level Features These include the number of mappers and reducers, the input file format, the compression method, etc. While these features do not distinguish between sibling tasks of the same job, they help in characterizing similar tasks across multiple jobs.

Data Skew Partitioning data over a low entropy key space might lead to a skew in the input sizes of tasks. In turn, this imbalance can cause a skew among the output sizes. Outlier input and output bulks are major contributors to task slowdown.

When the input size is not available at the moment of scheduling (e.g., for reducers), an estimate of it can be derived from a linear projection of the output produced by map tasks that have already completed. The actual size – and hence the data skew – becomes known at the end of the shuffle phase. The task's output size (which is never known in advance) can be linearly extrapolated in the middle of execution from the partial output produced and the fraction of input processed.

3.2 Hardware Features

Hardware features capture the state of all resources involved in distributed computation – nodes, disks, network, etc. Part of these features are constant (e.g., the node's hardware generation, in heterogeneous clusters), the others are fixed throughout the task's execution (e.g., the mapper's locality wrt data), and the rest are dynamic (e.g., performance counters). The latter can be roughly categorized into:

Node features reflect the local execution speed. They include CPU rate, memory usage, disk busy rate, I/O bandwidth, network bandwidth, number of TCP connections, number of JVM threads, number of local and remote data accesses, etc.

Network features reflect the data transfer rates (and possibly, interconnect bottlenecks). In this context, we use the following indicators: (1) intra-rack traffic, per rack; (2) cross-rack (upstream and downstream) traffic, per rack, and (3) cumulative traffic across the backbone.

Map-reduce features capture the MR-level information – e.g., the number of occupied slots, per node.

Alongside the instantaneous dynamic metrics, we also employ their aggregates over time, to capture the persistent trends – e.g., a sustained demand for RAM on some node. Specifically, we aggregates over two time windows – one 5-minute-long, and one 30-minute-long.

4 Machine Learning Technique

A predictor oracle receives a vector of task and node features, and returns a (real-valued) slowdown estimate. Our oracle implementation employs the popular *gradient-boosted decision tree* (GBDT) algorithm [3]. GBDT is an additive regression model comprised of an ensemble of binary decision trees. Each decision tree node is a split on some feature at a specific value, with a branch for each of the possible outcomes. Each terminal (leaf) node contains a score, which corresponds to the decision path. The resulting prediction is the sum of scores returned by individual decision trees.

The GBDT model minimizes the raw mean-square error (RMSE) among the labeled training samples (i.e., the historical tasks with computed slowdown values). We configure the training with the following parameters: the number of trees is 100, the number of leaves per tree is 10, and the shrinkage (or learning rate, [3]) is 0.5.

5 Empirical Evaluation

We build and evaluate separate models for mappers and reducers, due two to very different execution patterns of these task types.

We use a performance dataset collected from traces of a production Hadoop cluster at Yahoo! [4]. Two separate months of data are used for the train and test stages. Some cleaning steps are performed before applying the GBDT training mechanism on the data. We only learn from jobs with robust median task duration statistics, such that the slowdown values are meaningful. Each job in the training set has at least 20 mappers or reducers (depending on the target model). Furthermore, we only select jobs in which a "critical mass" of tasks has similar input sizes (unimodal distribution). Namely, at least 75% of the tasks must have input sizes varying from 90% to 110% of the median size. Finally, we filter out the jobs in which the median task input size is too small (4 MB or less), to filter out non-data-intensive jobs that exploit Hadoop for parallelism (e.g., CPU-bound simulations).

The refined training set contains approximately 1.2M data points (task instances). We use 200K uniform samples (mapper and reducer tasks) for training the respective models, and 100K samples for testing them.

We evaluated the predictor's accuracy with *coefficient of determination*, or (R^2) [5] – a statistical measure that stands for the fraction of variability in a dataset that is explained by the model. $R^2 = 1$ indicates a perfect fit (i.e., the predictor forecasts the test value without error), whereas $R^2 = 0$ stands for a total lack of correlation.

Figure 3 depicts the correlation between the predicted and actual task slowdowns, in log-scale, for a uniformly-sampled subset of tasks. A positive (resp., negative) value means that the task is slower (resp., faster) than the median similar task. Figure 3(a) stands for the mapper model, whereas Figure 3(b) stands for the reducer model. In this context, the points far above the diagonal are false positives (predicted slowdown too big), and those far below it are false negatives (predicted slowdown too small). The most valuable is the prediction accuracy for the rightmost points (real stragglers).

The prediction for mappers is very accurate (R^2 is almost 0.8), although some "heavy" slowdowns are mispredicted. The reducer oracle is much weaker (R^2 is approximately 0.4). Note that visually, in the former case most of the points are clustered around the diagonal, whereas in the latter they are more dispersed. The lower precision of the reducer model probably stems from the fact that we are trying to predict the overall shuffle and execution time, which is hard to capture due to different natures of these two processes. We conjecture that predicting the running time of each part independently could lead to better results.

A by-product of the learning process is the ranking of the model's features by their impact of the predicted function. We observe that the unbalanced task input and output are the primary reason for excessive slowdown (≥ 4) for about 70% of the mappers and only 40% of the reducers. Moreover, for the big jobs the big slowdown values, the ratio of straggler instances caused by reasons unrelated to data skew grows rapidly. The implication is that it is convenient to design the scheduling solution in a two-tiered fashion. The tasks with larger inputs (primarily mappers) should be scheduled first, independently from the other optimizations. The rest of the features should be handled separately, both in the scheduling and the speculative execution scenarios.

One more interesting observation is that the node's hardware generation is one of the most important root causes for slowdown. This fact that is often overlooked in heterogeneous clusters, in which all nodes are configured with the same number of map and reduce slots.

While these initial results are far from perfect, we hope to improve the prediction quality by exploiting the nature of workloads in production map-reduce clusters. In these environments, many jobs run in periodic fashion, often many times a day. Moreover, there is high correlation between the jobs' scale and recurrence – e.g., Table 1 shows that more than 95% of mappers and reducers belong to jobs that ran at least 50 times in 4 months.

In future work, prediction can be improved by (1) using a more refined definition of the task running time, as well as (2) employing some new relevant features. In the extreme case, specific models can be even tailored for the few dominant repetitive workloads.

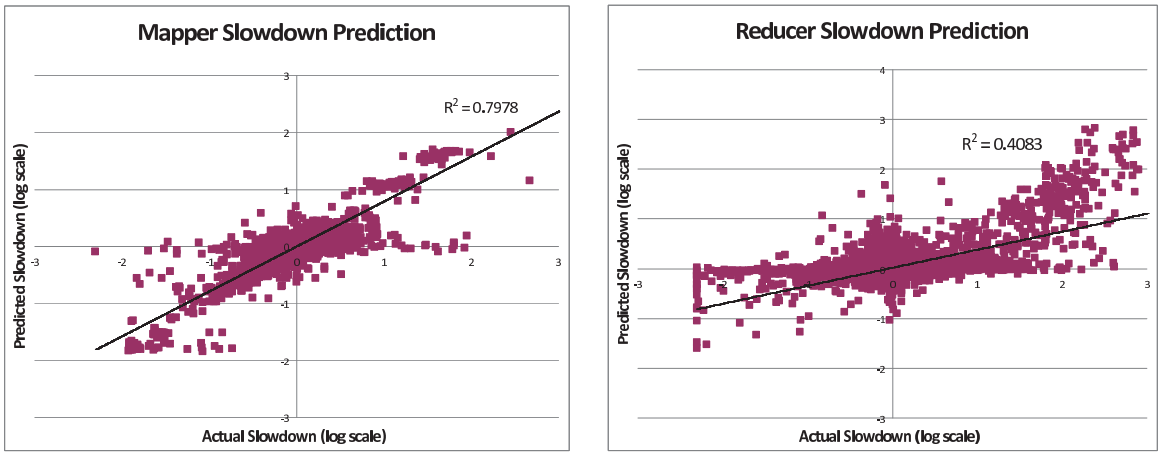

Figure 3: **The performance of slowdown predictors (predicted versus actual values), in log-scale. The prediction's quality is expressed by the statistical R^2 metric, which captures the variance explained by the model. The prediction for mappers is significantly more accurate ($R^2 = 0.79$) than for reducers ($R^2 = 0.41$).**

Recurrence (4 months)	Fraction of jobs	Fraction of mappers	Fraction of reducers
1	3.3%	0.1%	¡0.1%
2	2%	0.2%	0.1%
3-5	11.4%	0.3%	0.5%
6-10	6.2%	0.8%	0.5%
11-20	7.9%	1.1%	1.0%
21-50	10.9%	2.6%	2.7%
51-100	11%	2.9%	7.2%
101-200	32.9%	6.2%	9.2%
201-500	4.5%	9.8%	5.2%
501-1000	1.3%	5.6%	2.0%
> 1000	8.6%	**68.5**%	**71.5**%

Table 1: **Recurrence of workload in a production Hadoop cluster (4-month performance log). The vast majority of mappers and reducers belong to highly recurrent jobs.**

6 Related Work

Straggler task handling has been identified as a key issue in MR systems since the latter started maturing. The data-locality and speculative execution heuristics have been first implemented in Google's MapReduce [2]. The Hadoop infrastructure [6] introduced rack-awareness in placing data replicas and map tasks. These techniques reduced the straggler effect but failed to eliminate it. The Mantri research [1], which was used to optimize Microsoft's MR system, suggested new optimizations, e.g., network-aware task placement and early detection of stragglers, reporting a significant reduction in job latencies. However, the heuristic nature of these methods complicates tuning for diverse workloads.

Zaharia et. al. [7, 8] studied fairness issues in Hadoop scheduling that arise in heterogeneous environments, and showed the tension between fairness and locality. They evaluated improvements to Hadoop's scheduling policies in relatively small clusters. While these works are mostly orthogonal to ours, their results are in concert with our claim that over-optimizing data locality in MR systems might be counterproductive.

To the best of our knowledge, Kavulya et. al. [4] were the first to apply machine-learning methods to Hadoop logs. Their study presented multiple statistics about the utilization and workload in production MR clusters. They experimented with machine learning to predict the job execution times. However, the job-level granularity is too high to produce meaningful results for practical scheduling optimizations. We improve their ideas by introducing the task slowdown metric, which is amenable for applied use. Similarly to our work, Zhang et. al. [9] characterize production workloads in Web-scale MR clusters. They propose a simple way for generating synthetic map-reduce benchmarks, which can be used for evaluating our approach in practice.

7 Conclusions and Future Work

We presented *slowdown predictor* – a novel machine-learned component for bottleneck avoidance and detection in map-reduce (MR) systems. Slowdown prediction models can be integrated in scheduling and monitoring mechanisms of MR infrastructure implementations. These models can be trained on system performance traces that are routinely collected in production clusters. We believe that similarly to other domains, they can be tuned to specific workloads and hardware configurations faster and better than heuristic code.

Our initial evaluation shows that the new approach is viable, especially for mapper slowdown prediction. Yet, the prediction quality must be improved. In future work,

we will focus on better identifying the outlier slowdown instances (which affect the job latency most), rather than simply minimizing the average-case error metric.

We are working on a prototype implementation of a slowdown predictor for Apache Hadoop, to evaluate the new paradigm's benefits in practice.

Acknowlegement

We thank Ronny Lempel, Sriguru Chakravarthi, Amar Kamat and Mahadevan Iyer for stimulating discussions.

References

[1] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *ACM OSDI*, 2010.

[2] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, Jan. 2008.

[3] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[4] S. Kavulya, J. Tany, R. Gandhi, and P. Narasimhan. An analysis of traces from a production mapreduce cluster. In *IEEE/ACM CCGrid*, 2010.

[5] R. G. D. Steel and J. H. Torrie. *Principles and Procedures of Statistics*. McGraw-Hill, 1960.

[6] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media / Yahoo Press, 2 edition, 2010.

[7] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Eurosys*, 2010.

[8] M. Zaharia, A. Kowinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *ACM OSDI*, 2008.

[9] Q. Zhang, J. Hellerstein, and R. Boutaba. Characterizing task usage shapes in google compute clusters. In *LADIS*, 2011.