Optimization Strategies for A/B Testing on HADOOP

Andrii Cherniak
University of Pittsburgh
135 N Bellefield ave
Pittsburgh, PA 15260
aic3@pitt.edu

Huma Zaidi
eBay Inc.
2065 Hamilton ave.
San Jose, CA 95125
hzaidi@ebay.com

Vladimir Zadorozhny
University of Pittsburgh
135 N Bellefield ave
Pittsburgh, PA 15260
vladimir@sis.pitt.edu

ABSTRACT

In this work, we present a set of techniques that considerably improve the performance of executing concurrent MapReduce jobs. Our proposed solution relies on proper resource allocation for concurrent Hive jobs based on data dependency, inter-query optimization and modeling of Hadoop cluster load. To the best of our knowledge, this is the first work towards Hive/MapReduce job optimization which takes Hadoop cluster load into consideration.

We perform an experimental study that demonstrates 233% reduction in execution time for concurrent vs sequential execution schema. We report up to 40% extra reduction in execution time for concurrent job execution after resource usage optimization.

The results reported in this paper were obtained in a pilot project to assess the feasibility of migrating A/B testing from Teradata + SAS analytics infrastructure to Hadoop. This work was performed on eBay production Hadoop cluster.

1. INTRODUCTION

Big Data challenges involve various aspects of large-scale data utilization. Addressing this challenge requires advanced methods and tools to capture, manage, and process the data within a tolerable time interval. This challenge is trifold: it involves data volume increase, accelerated growth rate, and increase in data diversity [23]. Ability to perform efficient big data analysis is crucial for successful operations of large enterprises.

A common way to deal with big data analytics is to set up a pipeline of a high-performance data warehouse (e.g., Teradata [12] or Vertica [13]), an efficient analytics engine (e.g., SAS [9] or SPSS [7]) and an advanced visualization tool (e.g., MicroStrategy [8] or Tableau [11]). However, the cost of such infrastructure may be considerable [14].

Meanwhile, not every data analytics task is time-critical or requires the full functionality of a high-performance data analysis infrastructure. Often for non-time-critical applications, it makes sense to use other computational architectures, such as Hadoop/MapReduce. One might name Amazon[1], Google[33], Yahoo[15], Netflix[28], Facebook[35], Twitter[24], which use MapReduce computing paradigm for big data analytics and large-scale machine learning.

A/B testing [22] is a mantra at eBay to verify the performance of each new feature introduced on the web site. We may need to run hundreds of concurrent A/B tests analyzing billions of records in each test. Since A/B tests are typically scheduled on a weekly basis and are not required to provide results in real-time, they are good candidates for migration from the expensive conventional platform to a more affordable architecture, such as Hadoop [4]. This approach avoids the need for continuous extension of expensive analytics infrastructure. While providing less expensive computational resources, a corporate Hadoop cluster has impressive, but still finite computational capabilities. The total amount of the resources in the cluster is fixed once the cluster setup is complete. Each job submitted to a Hadoop cluster needs to be optimized to be able to effectively use those limited resources. One way is to use as many resources as possible in the expectation to decrease the time for execution of a job. However, hundreds of A/B tests need to be run concurrently, and Hadoop resources need to be shared between those jobs. Even a single A/B test may require as many as 10-20 MapReduce jobs to be executed at once. Each of these jobs may need to process terabytes of data, and thus even a single A/B test can introduce substantial cluster load. Some jobs may be dependent on other jobs. Thus for optimization purposes, we need to consider all jobs which belong to the same A/B test as a whole, not as independent processes. In addition, each job has to co-exist with other jobs on the cluster and not compete for **unnecessary resources**.

The results reported in this paper were obtained in a pilot project to assess the feasibility of migrating A/B testing from Teradata + SAS analytics infrastructure to Hadoop. Preliminary work was conducted at eBay in the Fall 2011. A month-long A/B test experiment execution and cluster resource monitoring was completed in the Fall 2012. All our experiments were executed on Ares Hadoop cluster at eBay, which in spring 2012 had 1008 nodes, 4000+ CPU cores, 24000 vCPUs, 18 PB disk storage [25]. The cluster uses **capacity scheduler**. All our analytics jobs were implemented using **Apache Hive**.

While performing data analytics migration, we tried to answer two questions:

Proceedings of the VLDB Endowment, Vol. 6, No. 11
Copyright 2013 VLDB Endowment 2150-8097/13/09... $ 10.00.

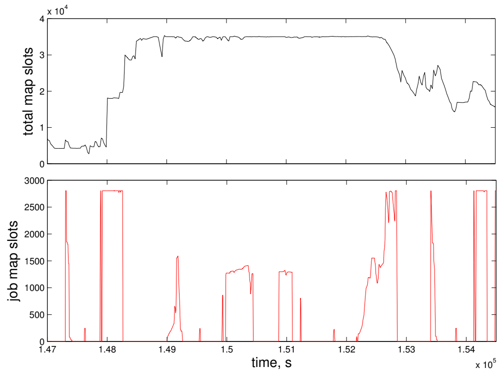


Figure 1: A/B test execution monitoring. Top plot: map slot usage in the entire cluster. Bottom plot: map slot usage by the A/B test jobs

- how to minimize the execution time of a typical A/B test on Hadoop;
- how to optimize resource usage for each job thus our A/B test can co-exist with other tasks;

Consider an example Hive job, repetitively executed on Hadoop, as shown in **Figure 1**. We monitored the amount of resources (here - the number of map slots) used by the job together with total map slot usage in the entire Hadoop cluster. The upper plot shows how many map slots were in use in the entire Hadoop cluster during the experiment. The bottom plot shows how many map slots our sample MapReduce job received during the execution. We observe that when the cluster is becoming busy, MapReduce jobs have difficulty accessing desired amount of resources. There are three major contributions we provided in this paper.

1. we provide empirical evidence that each MapReduce job execution is impacted with the load on the Hadoop cluster, and this load has to be taken into consideration for job optimization purposes
2. based on our observations, we propose a probabilistic extension for MapReduce cost model
3. we provide an algorithm for optimization of concurrent Hive/MapReduce jobs using this probabilistic cost model

Our paper is organized as follows. We start with providing some background on performance of Teradata data warehouse infrastructure and Hadoop in **Section 2**. We continue by presenting updated MapReduce cost model in **Section 3**. Then finally we apply this cost model to optimize a real A/B test running on a production Hadoop cluster and report the results in **Section 4**.

2. BACKGROUND

Dealing with big data analysis requires an understanding of the underlying information infrastructure. This is crucial for proper assessment of the performance of analytical processing. In this section, we start with an overview of Hadoop, - a general-purpose framework for data-intensive distributed applications. We also discuss the existing methods for data analytics optimization on Hadoop.

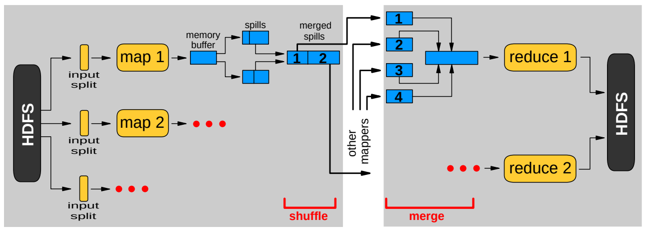


Figure 2: MapReduce execution schema

Apache Hadoop is a general-purpose framework for distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from a single server to thousands of machines, each offering local computation and storage. Hadoop provides the tools for distributed data storage (HDFS: Hadoop distributed file system [29]) and data processing (MapReduce). Each task submitted to a Hadoop cluster is executed in the form of a MapReduce job [16], as shown in **Figure 2**. JobTracker [4] is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster. The JobTracker submits the work to the chosen TaskTracker nodes. TaskTracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a JobTracker. A TaskTracker is configured with a set of slots (map and reduce), which indicate the number of tasks that it can accept [41] at a time. It is up to the scheduler to distribute those resources between MapReduce jobs.

There are several ways to write MapReduce jobs. The most straight-forward one is to write a Java program using MapReduce API. While this method provides the highest data manipulation flexibility, it is the most difficult and error-prone. Other tools, such as **functional languages** (Scala [10]), **data processing workflow** (Cascading [6]), and **data analysis languages** (Pig [3], Hive [35]) help to cover the underlying details of MapReduce programming, which can speed up development and eliminate typical errors. **Apache Hive** was chosen as a tool for these experiments because of its similarity with SQL, and to eliminate the need for low-level programming of MapReduce jobs.

2.1 MapReduce data flow

Every MapReduce job takes a set of files on HDFS as its input and, by default, generates another set of files as the output. Hadoop framework divides the input to a MapReduce job into fixed-size pieces called splits. Hadoop creates one map task for each split [5]. Hadoop takes care of scheduling, monitoring and rescheduling MapReduce jobs. It will try to execute map processes as close as possible [29] to the data nodes, which hold the data blocks for the input files. We cannot explicitly control the number of map tasks [5]. The actual number of map tasks will be proportional to the number of HDFS blocks of the input files. It is up to the scheduler to determine how many map and reduce slots are needed for each job.

2.1.1 Capacity scheduler

Our Hadoop cluster was configured to use capacity scheduler [2]. The whole cluster was divided into a set of queues with their configured map and reduce slots capacities. Hard capacity limits specify the minimum number of slots each queue will provide to the jobs. Soft limits specify how much extra capacity this queue can take from the cluster provided so that the cluster resources are under-utilized. When the cluster gets fully loaded, those extra resources will be reclaimed for the appropriate queues. In **Figure 1**, we observe the effect of this reclamation: when the total load on the cluster reached its maximum: MapReduce jobs in our queue were not able to obtain as many resources as they had before. A similar effect happens when we submit yet another job to a queue: each job from that queue will receive less resources compared to what they had before.

2.2 Current approaches for Hadoop / MapReduce optimization

We can group existing efforts in MapReduce jobs optimization into the following categories: scheduler efficiency, Hadoop and MapReduce (**MR**) system parameters optimization, and MR execution modeling.

2.2.1 Hadoop scheduler optimization

The default Hadoop scheduler is FIFO [41]. However, this may not be the best scheduler for certain tasks. The existing work in scheduling optimization addresses some of the most typical issues with execution optimization of MR jobs. One sample issue is resource sharing between MR jobs (Capacity scheduler [2], Fair Scheduler [45]), so one job does not suppress the execution of other jobs or does not occupy too many of the resources. [27] provides a summary of some of the existing schedulers for Hadoop. Here we provide a summary of the existing approaches to improve scheduling for Hadoop and how their optimization is applicable to our task of A/B testing.

Resource-aware scheduler [44] suggests using fine-granularity scheduling through monitoring resources usage (CPU, disk, or network) by each MR job. Delay scheduling [46] is trying to address the issue of data locality for MR job execution. Coupling Scheduler [31] approach is aimed at reducing the burstiness of MR jobs by gradually launching Reduce tasks as the data from the map part of a MR job becomes available. These schedulers treat each MR job as completely independent of one another, which is not true for our case. In addition, these schedulers say nothing about how to redistribute Hadoop resources between multiple concurrent MR jobs.

[38] introduced the "earliest-deadline-first" approach for scheduling MR jobs. The proposed approach (SLO-based scheduler) provides sufficient resources to a process, thus, it can be finished by the specified deadline. The authors report in the paper that when the cluster runs out of resources, then the scheduler cannot provide any guarantees regardless of process execution time. This happens because SLO-based scheduler is backed by FIFO scheduler [47]. Thus, first, we still need to perform off-line calculations about the optimal timing for each MR task. [38] says nothing about how to undertake this optimization, when there are many interdependent MR jobs, each of which is big enough to use all of the cluster resources available. Second, this approach is based on FIFO scheduler, therefore it provides the capability to control resources for each process, but this is not available for capacity scheduler which is running on our Hadoop cluster. And third, this approach does not consider the background Hadoop load caused by other processes, which can be launched at any arbitrary time.

[37] considers optimization of a group of independent concurrent MR jobs, using FIFO scheduler with no background MR jobs running (from other users) on the cluster. We use capacity scheduler instead of FIFO scheduler on our cluster, and we cannot explicitly control resources assignment for each MR job. Typically A/B testing jobs show strong dependency between each other, and this dependence impacts the performance.

[42] presented a scheduler which tries to optimize multiple independent MR jobs with given priorities. It functions by providing requested amount of resources for jobs with higher priority, and redistributes the remaining "slack" resources to jobs with lower priorities. [42] provides batch optimization for a set of MR jobs where each job is assigned its priority. This is not true for our case: other users submit their jobs whenever they want and can take a significant portion of the available resources. Thus, the approach has to take on-line load into consideration.

2.2.2 Hadoop/MapReduce system parameter optimization

There were several approaches to model Hadoop performance with different levels of granularity. For instance, [40] considers a detailed simulation of the Hadoop cluster functionality. It requires the definition of a large number of system parameters; many of them may be not available to a data analyst. Similarly, [19] is based on quite detailed knowledge of the cluster set up. [21] and [20] approaches relax the need for up-front knowledge of your cluster system parameters and provide a visual approach to investigate bottlenecks in system performance and to tune the parameters.

While these approaches address very important issues of MR job and Hadoop cluster parameter tuning, they do not address a more subtle issue: concurrent job optimization for a cluster which is shared between many users, executing all sorts of MapReduce jobs. These approaches do not consider how to adjust Hadoop/MR parameters so that many MR jobs can effectively co-exist.

2.2.3 MapReduce cost model

While MR job profiling [20] can help to optimize a single job, it is much more practical to have a simplified generative model which would establish the relationship between the MR job execution time, the size of the input dataset and the amount of resources which a MR job receives. Thus, we can calibrate this model once by executing a few sample MR jobs, measure their execution time, and use the obtained model for a quick approximate prediction of a MR job completion time.

[37], [36], [39], [43], [19], [40] attempt to derive an analytical cost model to approximate MR execution timing. Each of these approaches specifies an elementary cost model for data processing steps in MR and then combine elementary cost estimates in an overall performance assessment. It is interesting to mention, that different approaches have differences even with this simplified model. For instance, [39] and [37] do not account for sorting-associated costs and the overhead required to launch map and reduce jobs as in [36]. These differences are associated with the fact that MR framework can launch in-memory or external sorting [41]. And thus, cost models obtained for different sizes of the data set will differ.

The Hadoop/MR framework has a substantial number of tuning parameters. Tuning some of those parameters may have a dramatic influence on the job performance, while tuning others may have a less significant effect. Moreover, some of the tuning may have a non-deterministic impact, if we optimize the usage of a shared resource. For instance, assume that after the job profiling from [21] we decided to increase the sort buffer size for a MR job. Let us do the same for every MR job. If, after this buffer tuning, we launch too many MR jobs at once, some nodes may run out of RAM and will start using the disk space for swapping to provide the requested buffer size. Thus, from time to time, instead of speeding up, we will slow down concurrent job execution. This non-deterministic collective behavior has to be reflected in the cost model.

We based our cost model on the approaches proposed in [36] and [43], to reflect the deterministic characteristics of MR execution. We extend those models by adding the probabilistic component of MR execution. In **Section 3**, we elaborate on the MapReduce cost model and describe how we use it to estimate our Hadoop cluster performance.

2.2.4 *Applicability limits of the existing solutions towards large-scale analytics tasks*

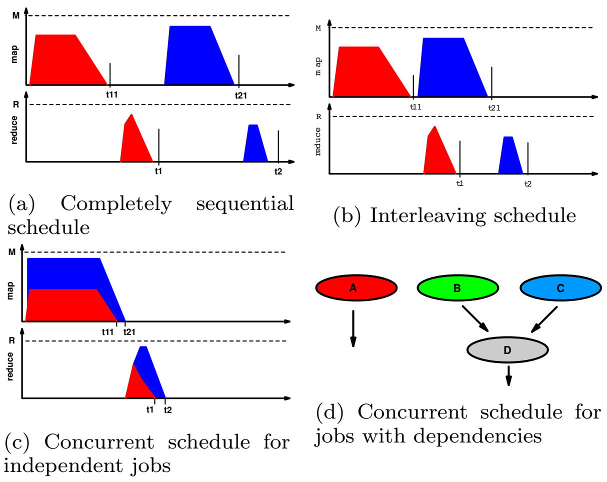
While the Hadoop community has made great strides towards Hadoop/MR optimization, those results are not exactly plug-and-play for our A/B testing problem. Our production cluster has capacity scheduler running and we cannot change that. Approaches assuming any other scheduler cannot be directly applied here. We cannot explicitly control the amount of resources for each MR job, because capacity scheduler controls resource sharing dynamically. We can only explicitly set the number of reduce tasks for a MR job. Our results were obtained for Hive using **speculative execution** model[41], thus Hadoop can launch more reduce tasks than we specified. Production analytics jobs need to be executed on a real cluster which is shared between many people who submit their jobs at arbitrary times. Thus, any optimization strategy disregarding this external random factor is not directly applicable.

3. UPDATED MAP-REDUCE COST MODEL

3.1 Concurrent MapReduce optimization

Let us assume that we submit MapReduce jobs to a Hadoop cluster with a queue Q, which has limits: M - max. number of map slots and R - max. number of reduce slots. At first, let us have 2 **independent** MapReduce jobs (colored in red and blue). We will look at different scenarios for those jobs to execute. If we submit them sequentially, as shown in **Figure 3a**, then the cluster resources will remain idle for much of the time. When the map part for the red job is over, map slots will remain idle while Hadoop is processing the reduce part for the job. From **3a:**t_{11} to **3a:**t_1 map slots are idle.

The most obvious solution to improve resource utilization is to monitor the progress of running jobs and launch new jobs when the cluster is idle. **Figure 3b** demonstrates this principle. At time **3b:**t_{11}, when the map part of the red job is complete, the blue job gets launched. Thus, from **3b:**t_{11} to **3b:**t_1 the Hadoop cluster has its map and reduce slots utilized. This schedule will reduce the total execution time: **3b:**t_2 < **3a:**t_2, as shown in [48]. In [38], they report a map slot utilization diagram when running a simulation for



(a) Completely sequential schedule

(b) Interleaving schedule

(c) Concurrent schedule for independent jobs

(d) Concurrent schedule for jobs with dependencies

Figure 3: Examples of execution schedule for MapReduce jobs

MapReduce job execution. Map task execution may have long tails where most of the tasks have been completed, however the whole job is waiting for the last few map tasks. This situation is a variation of resource starvation problem [32].

Another possible improvement to the scheduling scenario shown in **Figure 3c**, is when we submit all independent jobs simultaneously. This approach will maximize map and reduce slot usage. However, if some of the jobs are interdependent, this approach may lead to a very interesting problem.

Consider an example scenario from **Figure 3d**. Here, task D consumes the output from tasks B and C. Task A is completely independent of tasks B,C, and D. The optimization in this scenario is difficult because it depends on all 4 tasks and how many resources provided by Hadoop. In a typical setup for an A/B test, **each of those four tasks is big enough that, on its own, it can occupy all map and reduce slots in the queue**. If we apply the optimization logic from **Figure 3c**, then task A will consume a portion of the map and reduce slots, thus tasks B and C will take longer to finish their work and task D will start (and finish) later. If task A is relatively quick, then we would want to implement optimization from **Figure 3b** when task A launches its map jobs after tasks B and C finished their map part and then execute the reducer jobs. If task A is relatively long, then we would like to use a scheduling scenario from **Figure 3c** when task A is launched together with B and C, but receives a lower priority and uses only spare map slots and does not slow down B and C. There are many possible scenarios in this 4-task problem, and the solution to those are often non-trivial. We need to consider that:

1. other users submit their MapReduce jobs to the cluster and each job which we submit receives less resources than it is asking for

2. there is no way to explicitly control the amount of resources for each MapReduce job when using Capacity Scheduler

Launching concurrent MapReduce jobs helps to utilize Hadoop resources more effectively. Yet each of those jobs

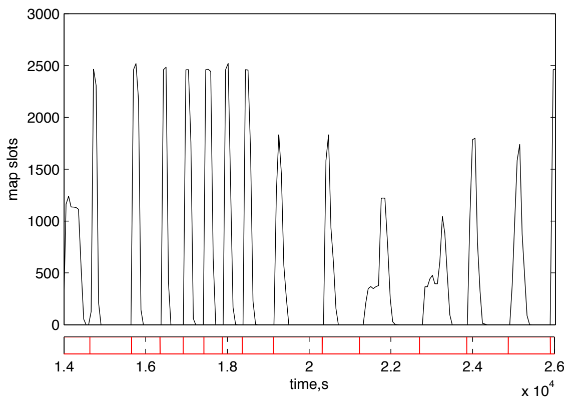


Figure 4: Upper plot: Map slots' usage for MR jobs. Lower plot: indicates boundaries for each MR job

may influence other jobs running on the cluster. The scenario presented in **Figure 3d** is a very typical one for analytics jobs. Thus resource optimization needs to consider influence from other jobs. In the following subsection we will address this issue.

3.2 Cost model based on probabilistic resource allocation

Assume that we have a MR job, which takes M blocks of data as its input and a user specified R reduce tasks to be executed. We have m slots available to run map tasks and r slots to run reduce tasks. Using the notation from [36], to complete the map phase we need $\lceil\frac{M}{m}\rceil$ rounds to process it, since in one cycle Hadoop can execute only m map tasks. Following the same reasoning, it takes $\lceil\frac{R}{r}\rceil$ cycles to complete the reduce part and the total MR execution time becomes **Equation 1**:

$$T = \lceil\frac{M}{m}\rceil F_m + \lceil\frac{R}{r}\rceil F_r + \Theta(M,R) \tag{1}$$

where F_m is the time required for one map task to complete, F_r is the time required for one reduce task to complete, and $\Theta(M,R)$ -is the cost associated with the overhead of launching map and reduce tasks. We are interested in processing a significant amount of data, thus $M >> m$. For practical purposes, we can replace

$$\lceil\frac{M}{m}\rceil \approx \frac{M}{m} \tag{2}$$

Figure 4 shows how many map slots were assigned to the same MR job over time when it was recursively executed. During a MR job execution, resource usage is not constant and depends on how many resources are available. Thus, we should replace parameter m with its effective value, shown in **Equation 3**

$$\lceil\frac{M}{m}\rceil \approx \frac{M}{\frac{1}{N}\sum\limits_{m_i\in m_+} m_i} = \frac{M}{I_m}; \quad ||m_+|| = N \tag{3}$$

where m_+ is a set of measurements during a single MR job when map slot usage by this MR job was > 0. Similar reasoning goes for $\lceil\frac{R}{r}\rceil$. Consider **Figure 5** which shows a part of the experiment on reduce slots' usage. In this experiment, we sequentially executed the same Hive job, asking for different numbers of reduce tasks to be executed:

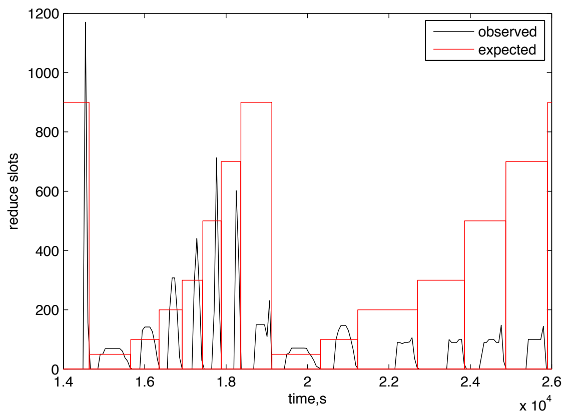


Figure 5: Reduce slots' usage as a function of Hadoop cluster load and speculative execution

[**50, 100, 200, 300, 500, 700, 900**]. The red line on **Figure 5** shows the number of reducers we would expect to be executed for the job. The black line shows the actual number of reducers which were executed. We see that the number of reducers is not constant: it changes during the reduce part. One of the reasons is that the cluster may be busy during the job execution and some of the reducers will be reclaimed back to the pool.

We observe examples when we aimed to use 900 reducers but received only about 100. It also happens that Hadoop launches more reduce tasks than we asked for. This happens because Hadoop detects that some of the reducers made less progress than the majority, and launches copies of slow running reducers.

Based on this reasoning, we transform $\lceil\frac{R}{r}\rceil$ into **Equation 4**. Here the denominator is the area under the reduce slots usage curve. To eliminate counting of extra reducers (from speculative execution), we limit the maximum number of reduce slots to R. For example, if we requested 300 slots but at some point we got 350 slots, we would count only 300.

$$\lceil\frac{R}{r}\rceil \approx \frac{R}{\frac{1}{N}\sum\limits_{r_i\in r_+} min(r_i,R)} = \frac{R}{I_r}; \quad ||r_+|| = N \tag{4}$$

where r_+ is a set of measurements during a single MR job where reduce slot usage by this MR job was > 0. Combining **Equations 1,2, and 4**, we obtain **Equation 5**

$$T = \beta_0 + \beta_1\frac{M}{I_m} + \\ + \frac{R}{I_r}\left(\beta_2\frac{kM}{R} + \beta_3\frac{kM}{R}log(\frac{kM}{R})\right) + \beta_4 M + \beta_5 R \tag{5}$$

In **Equation 5** β_0 is only a constant; $\beta_1 M_m$ is the time required to read the MR job input; kM is the size of the output of the map part of the MR job, $k \geq 0$; $\beta_2\frac{kM}{R}$ is the time needed to copy the data by each reducer; $\beta_3\frac{kM}{R}log(\frac{kM}{R})$ is the time required to merge-sort data in each reducer; $\beta_4 M + \beta_5 R$ is the cost associated with the overhead of launching map and reduce tasks, as suggested in [36]. However, from the dynamics of map and reduce tasks shown in **Figure 4** and **Figure 5**, it is not clear which number should be put in the calculation. We will assume that this extra cost is relatively small compared to the timing caused by the actual

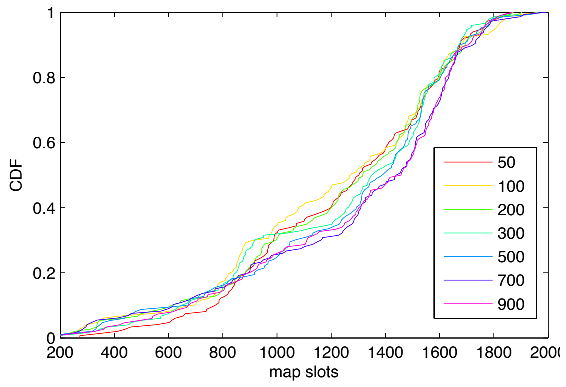


Figure 6: CDF plot for map slots' allocation to a MR job as a function of requested reducers

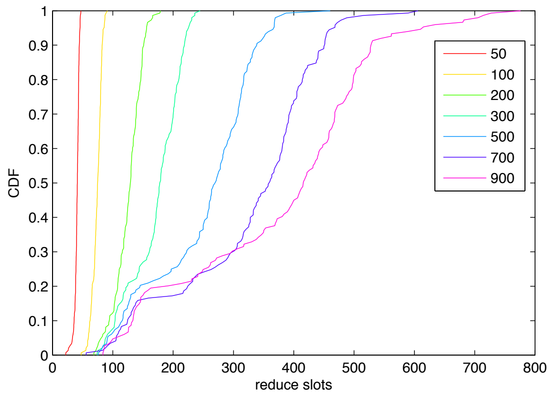


Figure 7: CDF plot for reduce slots' allocation to a MR job as a function of requested reducers

data processing, and will omit this overhead part, like, for instance, in [38]. We obtain **Equation 6**.

$$T = \beta_0 + \beta_1 \frac{M}{I_m} + \beta_2 \frac{kM}{I_r} + \beta_3 \frac{kM}{I_r} log(\frac{kM}{R}) \tag{6}$$

3.3 Probabilistic resource allocation

Job completion time in **Equation 6** depends on how many resources a MR will receive. We conducted 150 iterations of the same cycle of MR jobs, each cycle consists of 7 sequential MR jobs, asking for **[50, 100, 200, 300, 500, 700, 900]** reducers, totaling in 1050 jobs execution and 9-day duration.

Resource allocation for MR jobs are shown in: **Figure 6** for map slots allocation and **Figure 7** for reduce slots allocation. From **Figure 6**, we observe that in probability, each MR job received the same number of map slots. This happens because we cannot explicitly control map slot assignment and must rely on the scheduler to obtain resources. The situation is completely different with the reduce slots: we can explicitly ask for a specific amount of the resources. However, even if we ask for more slots, it does not mean that we will receive them. If we ask for fewer resources, most likely the scheduler will be able to provide those to a MR job. In **Figure 7**, when we ask for 50 reduce slots, in 90% of cases we will receive those 50 slots. However, when we ask for 900 reducers, in 90% of cases we will receive less than 550 reducers. From **Equation 6**, the volatility of the

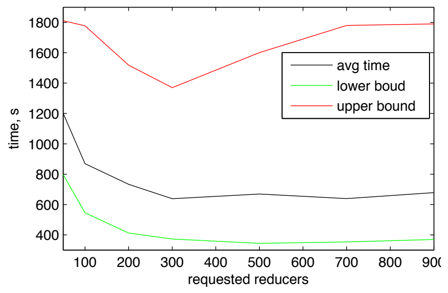


Figure 8: MapReduce job completion time as a function of requested reduce slots: (average time, upper and lower bounds for 95% interval)

MR job completion time will increase when we ask for more reduce slots.

Figure 8 provides another perspective on the volatility of the MR execution. We have plotted the average completion time of a MR job as a function of the number of requested reduce slots together with 95% interval of the observed measurements. We observe that for jobs requesting 300 reducers or more, the average completion time remains almost constant. However the 95% upper bound of the execution time is increasing. This time increase is connected to the dynamic nature of Hadoop cluster load: some of the nodes which execute our MR job reduce tasks may arbitrarily receive extra load (other users submit their MR jobs). Speculative execution [41] was introduced to Hadoop to mitigate this problem, however it does not solve the issue completely. The higher the number of the reducers requested, the higher the chances are that at least one reducer will get stuck on a busy node. To reflect this effect we propose to alter the MapReduce cost model for **Equation 6** into **Equation 7**.

$$T = \beta_0 + \beta_1 \frac{M}{I_m} + \left(\beta_2 \frac{kM}{I_r} + \beta_3 \frac{kM}{I_r} log(\frac{kM}{R}) \right) * (1 + D_R(R)) \tag{7}$$

where $D_R(R)$ is probabilistic extra delay, associated with the chance that a reducer gets stuck on a busy node. For each R, $D_R(R)$ is a set, containing pairs (delay, p) - shows possible extra delay together with the probability to observe this extra delay. At first, we assume that $D_R(R) = 0$ and obtain the coefficients β for **Equation 7**. Then we add [70..95] interval of all measurements and learn that $D_R(R)$.

3.4 Functional dependencies for resource allocation

Figure 9 shows how map resources were assigned to a local queue as a function of the total cluster map slot usage. We observe that the distribution of the measurements does not change significantly until the total cluster load reaches approximately $3{*}10^4$ map slots. Below that threshold, queue load and total cluster load seem to be uncorrelated, e.g.: $p(Q|HC) = p(Q)$, where $p(Q)$ is the probability to observe certain queue load, and $p(HC)$ is the probability to observe certain total Hadoop cluster load. When the cluster load becomes higher than $3 * 10^4$ in **Figure 9**, we can clearly see that the total cluster load influences the number of slots which a queue will get: $Q \leftarrow HC$, and $p(Q|HC) \neq p(Q)$.

Obviously, the amount of the resources which a MR job obtains depends on how many resources in a queue are used

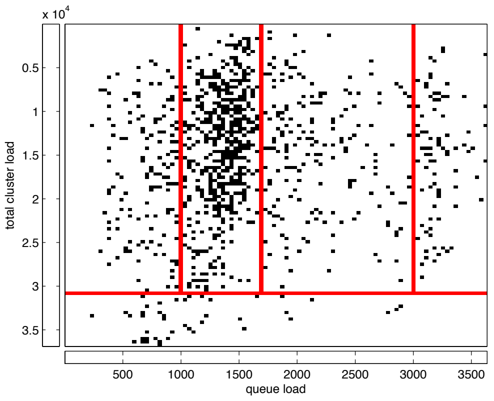
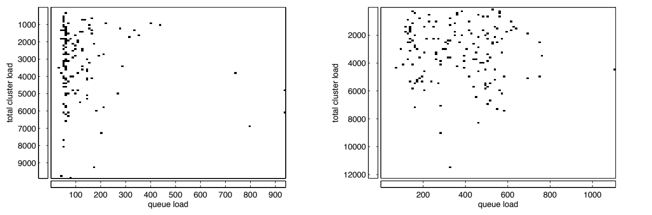


Figure 9: Map slot usage in a queue as a function of total cluster load



(a) here 50 reducers were requested (b) here 900 reducers were requested

Figure 10: Reduce slot usage in a queue as a function of total cluster load and number of requested reducers

by other jobs, and whether map slots can be borrowed from the rest of the cluster. If we denote r_J to be the amount of resources for a job J, then $r_j \leftarrow HC, Q$. For given values of hc_i - particular Hadoop load, and q_i - particular queue resource usage, $p(r_J) = p(r_J|q_i, hc_i)$.

Figure 9 is intrinsically 3D, where the 3rd dimension - the number of map slots which a MR job obtained, is collapsed. Hadoop load analysis provides us with **(Hadoop load, queue load, MR resources)** tuples, which allow us to build a probabilistic model to describe how many resources we would receive given what we know about the total cluster load and other jobs in the queue, as shown in **Equation 8**.

$$p(r_J) = \sum_{hc_i \in HC} \sum_{q_i \in Q} p(r_J|q_i, hc_i) * p(q_i|hc_i) * p(hc_i) \quad (8)$$

Figure 10a and **Figure 10b** show how reduce resources are being distributed in a queue as a function of the total reduce usage in a cluster. What we conclude is that the nature of the distribution changes depending how many reducers we are seeking. We can derive **Equation 9** for probabilistic reduce resource allocation, similar to **Equation 8**:

$$p(r_J, R) = \sum_{hc_i \in HC} \sum_{q_i \in Q} p(r_J|q_i, hc_i, R) * p(q_i|hc_i, R) * p(hc_i) \quad (9)$$

where parameter R - specifies how many reducers we actually asked for.

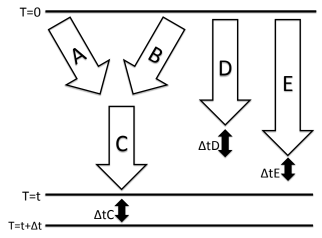


Figure 11: Resources sensitivity explained

3.5 Applying stochastic optimization for concurrent MapReduce jobs

Before we proceed to the algorithm description, we need to introduce the notion of resource sensitivity for a MR job. Assume that our task consists of a few MR jobs with certain data dependencies between them, e.g. **Figure 11**. Resource sensitivity shows how the task execution time changes if we reduce the amount of resources for a particular job J. From the example in **Figure 11**: the task starts at $T = 0$ and finishes at $T = t$. Now lets assume that we decrease the number of map or reduce slots for a particular job J on ΔR, which may lead to the increase of the task execution time from $T = t$ to $T = t + \Delta t$. We use **Equation 7** to predict a job completion time. Task resource sensitivity for a particular job J is shown in **Equation 10**.

$$RS(J) = \frac{\Delta t}{\Delta R} \quad (10)$$

In the example shown in **Figure 11**, $RS(D) = RS(E) = 0$ because the increase in the execution time for jobs D and E does not influence the total timing: job C finishes its execution later than jobs D and E (even after we reduce the amount of resources for jobs D and E). However $RS(C) > 0$, because job C finishes its execution last, and any resource reduction to job C increases the task total execution time.

The central idea for our stochastic optimization is to find those MR jobs, in which results are needed much later during the A/B test execution and which require fewer resources or can be assigned a lower execution priority. In capacity scheduler we cannot explicitly control map slot assignment to a job. However, when we set a lower priority to a MR job, we force this job to use map and reduce slots only when they are not used by other processes in the same queue from the same user. When there are many users submitting their jobs to a queue, capacity scheduler provides a share of a queue resources to each user. Thus, even the lowest priority Hive job will obtain a portion of the resources. All these considerations help us to derive **Algorithm 1** for MR job optimization strategies.

4. CASE STUDY: MIGRATING A/B TEST FROM TERADATA TO HADOOP

In this paper, we focus on one particular example of Big Data analytics: large-scale A/B testing. We performed an A/B test for eBay Shopping Cart dataset. An outline of the test is shown in **Table 1**. Originally this test was executed using both Teradata and SAS. Teradata would pre-process data and send the result to a SAS server, which would finalize the computations. We start with an overview of Teradata - a high-performance data warehousing infrastructure, and then move to the discussion of the A/B test schema.

```
Algorithm 1: Stochastic optimization for A/B test
input  : MR job dependence list; MR job data size;
         MR performance model Equation 7; Hadoop
         load model Equation 8, 9
output: reducer allocation per MR job in the task,
         priorities for each MR job
begin
    1: obtain pairs ($Resources_i; p_i$) - amount of the
    resources for the task with the probability to obtain
    those resources, using Equation 8, 9
    2: using $Resources_i$, instantiate each MR job with
    map and reduce resources, proportionally to the job
    input data size
    repeat
        repeat
            1: compute reducer resources sensitivity
            using Equation 10
            2: add $\Delta R$ of reducers to those jobs, which
            can help to decrease the task execution time
            if total timing does not increase then
                1: decrease the # of reducers on $\Delta R$ for
                those jobs with the lowest reducer
                resources sensitivity
                2: use Equation 7 to compute task
                total timing
            end
        until no improvement;
        foreach job in the test do
            1: using Equation 10, compute map
            resources sensitivity, as if the job was
            assigned lower priority
        end
        1: choose the job J with the lowest map
        resources sensitivity
        2: assume that job J to be set lower priority;
        recalculate map and reduce slot assignment to
        other MR jobs and total timing using Equation
        7
        if total timing did not increase then
            1: assign lower priority to job J
            2: recalculate map and reduce slot
            assignment to other MR jobs
        end
    until no improvement;
    1: repeat for each pair ($Resources_i, p_i$)
    2: report resources assignment for each job as the
    expectation of the resources assignment
end
```

4.1 Teradata

Teradata [12] is an example of a shared-nothing [30] massive parallel processing (MPP) relational database management system (RDBMS) [34], [26]. A simplified view of its architecture is shown in **Figure 12**. The key architecture components of Teradata are: **PE** - parsing engine; **AMP** - Access Module Processor; **BYNET** - communication between VPROCs.

AMP and PE are VPROCs - virtual processors, self-contained instances of the processor software. They are the basic units of parallelism [18]. VROCs run as multi-threaded processes to enable Teradata's parallel execution of tasks without using specialized physical processors. They are labeled as "Virtual Processors" because they perform the similar processing functions as physical processors (CPU). In Teradata architecture, each parallel unit (AMP) owns and manages its own portion of the database [26], [17]. Tuples are hash-partitioned and evenly distributed between AMPs. And the workload for joining, aggregates calculation and sorting can be achieved in a way that the load redistributes equally between AMPs. As with any RDBMS, Teradata uses indexing [34] on tables, which allows speeding up data access.

procedure	equivalent SQL query	engine
extraction	$A = t_1 \bowtie \cdots \bowtie t_5$	Tera-data
pruning	$A_1 =$ select $A.c_1, \ldots$ where ...	
aggregation	$A_2 =$ select $stddev(A_1.c_1), \ldots$ group by ...	
capping	Update A_1 set $A_1.c_1 =$ $= min(A_1.c_1, k * A_2.stdc_1), \ldots$	SAS
aggregation	$A_3 =$ select $stddev(A_1.c_1), \ldots$ group by ...	
lifs, CIs	computed using SAS	

Table 1: A/B test schema

table	Original number of tuples	Number of tuples after pruning
t_1	9,125	48
t_2	429,926	215,000
t_3	2,152,362,400	533,812,569
t_4	4,580,836,258	263,214,093
t_5	6,934,101,343	3,250,605,119

Table 2: Data set size

4.2 Test schema

While select, join and aggregate computations are the routine RDBMS operations, the real challenge is the size of data in tables $t_1, \ldots t_5$, which is shown in **Table 2** as the original number of tuples. In our sample A/B test, all the heavy data processing was executed on Teradata, and a much smaller data set would be sent to SAS. Thus, it is more important to compare the Teradata part of the A/B test with its Hadoop equivalent.

The Teradata part of the A/B test from **Table 1** can be translated 1-to-1 into Hive queries. The SAS part of the

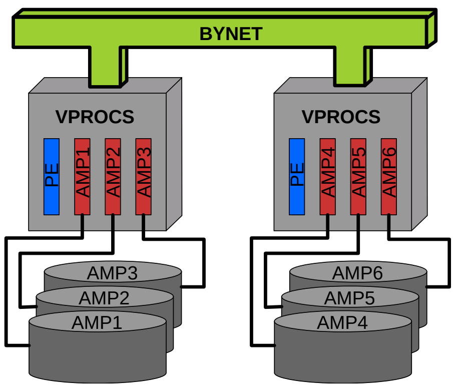
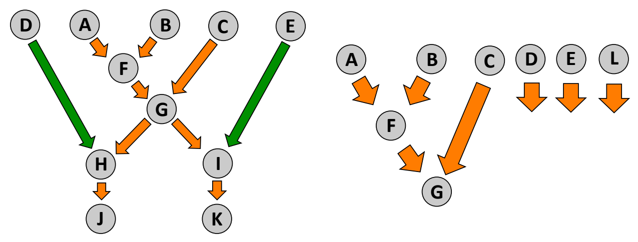


Figure 12: Teradata



(a) scenario 1 (b) scenario 2

Figure 13: A/B test scenarios:

$A = t_4 \bowtie \sigma(t_1)$ $B = \sigma(t_5)$ $C = \sigma(t_2)$ $D = \sigma_{t_3.c=v_3}(t_3)$ $E = \sigma_{t_3.c\neq v_3}(t_3)$
$F = A \bowtie B$, $G = F \bowtie C$, $H = G \bowtie D$, $I = G \bowtie E$
$J = sum(\dots), count(\dots)$ *from* H, $K = sum(\dots), count(\dots)$ *from* I

Table 3: A/B test data loading: extraction, pruning, and aggregation

A/B test cannot be translated completely into Hive. We use PHP scripting language to finalize the computations which cannot be translated in Hive. However, PHP scripts would perform only a very small portion of final data assembly.

The schema in **Table 1** possesses strong dependencies between different execution steps. For instance, we cannot proceed with capping unless we computed the aggregates of the dataset. Or, we cannot compute lifts and confidence intervals unless we capped the data.

After data dependency analysis, we transform the Teradata part from **Table 1** into the diagram, shown in **Figure 13a**. Letters A through K denote the data processing operations, as shown in **Table 3**. So now we can proceed with the comparison.

4.3 A/B test without explicit resource control

We run sequential and parallel versions of data loading routines from **Table 3** on Hadoop and compare the obtained results with Teradata timing. In these experiments, **we do not control the amount of resources (number of reduce tasks per Hive job). We let Hive to infer this based on the data size for each job**. However, all of the experiments were performed during the weekends, when the queue to which we were submitting MR jobs, was completely free. Each result was averaged over 10 executions.

In the first experiment, we execute data loading routines completely sequentially. In the second experiment, we schedule those routines concurrently, preserving the data dependencies between them, as shown in **Figure 13a**. We launch jobs **A, B, C, D, and E** simultaneously, and proceed with other jobs as the data becomes available.

We conducted experiments with different sizes of table t_3 for data loading on both Hadoop and Teradata. Data load timing for Hadoop is shown in **Figure 15**, and for Teradata is revealed in **Figure 14**. At first, we observe that sequential data loading on Hadoop takes approximately 70 minutes (**Figure 15**) when table t_3 contains 14 % of data. For concurrent Hadoop loading routines, it takes approximately 20 minutes to execute the routines having 100% of data in table t_3.

For Teradata, it takes only 2 minutes to load the data having 100% of the size of table t_3. However, the slope of the plot for Teradata in **Figure 14** is much steeper than for Hadoop in **Figure 14**. When the amount of the data in table t_3 increases, execution time for Teradata increases almost linearly. The reason is that data selection from the table happens in the background of running other processes. Processes D and E in **Figure 13a** select data from the table and their results are required only in later stages of job execution.

In **Figure 16**, we compare the total timing to compute an A/B test using Hadoop only with a combination of Teradata + SAS. **While Teradata is about 10 times faster than Hadoop to load the data, Teradata + SAS appears to be about 5 times slower to compute the whole A/B test, than to do it on Hadoop.**

4.4 Applying stochastic optimization for A/B test

For this evaluation, we will modify the data loading schema for the A/B test from **Figure 13a** into the one shown in **Figure 13b**. In this scenario, we added an extra job L which processes a substantial amount of data, but the results of which are needed only at the later stages of the A/B test. We would like to investigate how this extra job impacts the performance. To be able to use stochastic optimization, we need the corresponding data sizes for each of these jobs. Those numbers are shown in **Table 4**.

When we apply **Algorithm 1** to our modified A/B test as shown in **Figure 13b**, we note that jobs D, E, and L receive lower priority of execution. Also job L receives 150 reducer slots instead of 800 slots, originally determined by

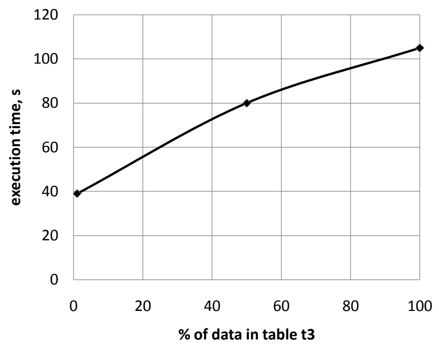


Figure 14: Timing for data extraction, pruning, and aggregation on Teradata

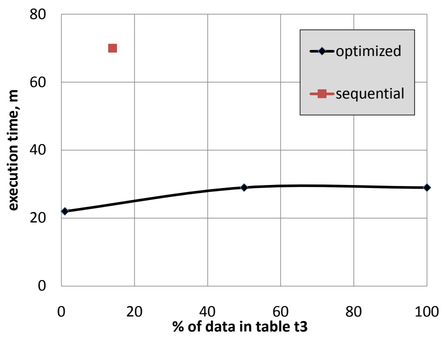


Figure 15: Timing for data extraction, pruning, and aggregation on Hadoop

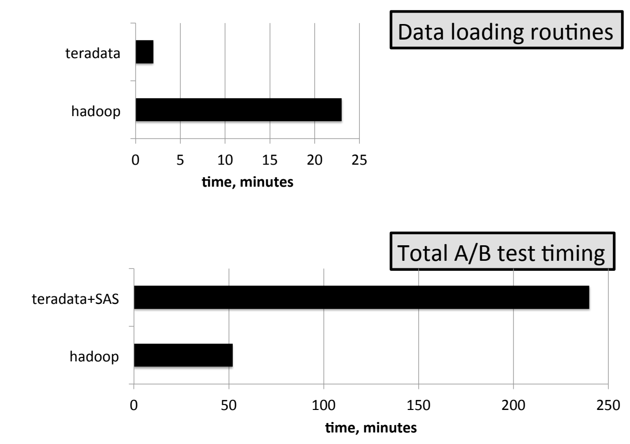


Figure 16: Timing for cart A/B test. **top:** time comparison of data loading routines, executed on Hadoop and Teradata; **bottom:** time comparison for execution of the whole A/B test on Hadoop vs Teradata+SAS

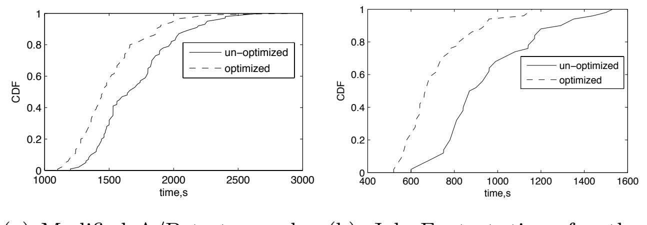
Job name	Input size (TB)	Output size (TB)
A	1.909100	0.002200
B	1.194800	0.201700
C	0.000014	0.000008
D	0.810000	0.004000
E	0.810000	0.006000
F	0.204000	0.002600
G	0.002600	0.002600
L	0.814800	0.002300

Table 4: MapReduce job size

Hive based on the size of the data input

4.5 Stochastic optimization results

Here we use the modified A/B test scenario from **Figure 13b**. In the first schema, we let Hive determine on its own about how many resources to provide for each job. In the second schema, we apply **Algorithm 1** to optimize the execution. The result of this stochastic optimization is shown in **Figure 17a**. The un-optimized A/B test was executed 100 times; the optimized version of A/B test was executed 120 times. The first 50 iterations of both the un-optimized and optimized tests were executed from Saturday night through Sunday. The remaining iterations were executed from Monday through Tuesday. In this way, we tried to measure the effect of stochastic optimization for an A/B test on variation in Hadoop cluster load.



(a) Modified A/B test completion time (b) Job F start time for the modified A/B test schema

Figure 17: CDF plots for:

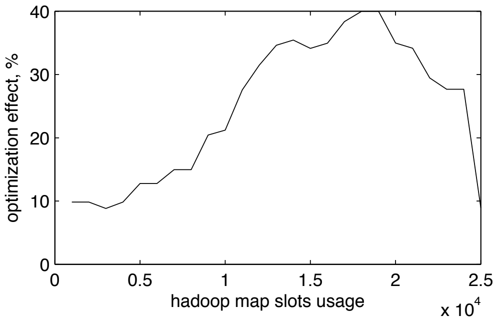


Figure 18: Optimization effect for a modified A/B test schema as a function of total Hadoop map slots usage

We observe that the optimized version of data loading may take take approximately 16% less time to complete the task execution. Here, we count the optimization effect as the biggest difference between two cumulative distribution functions **CDF** for both optimized and un-optimized results. We divide this difference on the corresponding timing for un-optimized schedule. This optimization effect happens because jobs D, E, and L from **Figure 13b** do not interfere with the main execution line. And therefore jobs A, B, F, and G receive as many resources as available or they possibly can utilize and speed up the execution. Consider the result from **Figure 17b**, which shows how the start time for job F changes for optimized and un-optimized scenarios. When jobs D, E, and L do not compete for resources with jobs A and B, then A and B finish earlier and job F can start earlier. Meanwhile, jobs D, E, and L obtain their resources when these are not in use by the higher-priority jobs.

Figure 17a displays a cumulative optimization effect. We would like to know the effect of this optimization, depending on the load on a Hadoop cluster, for both map and reduce slots usage. **Figure 18** shows the effect of optimization strategies as a function of map slot usage on the entire Hadoop cluster. Here Hadoop map slot usage was calculated only during times when the A/B test jobs were using map slots. The experimental results show **the biggest optimization effect for higher levels of cluster load**, which is a very valuable contribution of this optimization. The reported value of $2.5 * 10^4$ was the highest integral map slot usage on the cluster during our experiments.

A different optimization effect is observed regarding reduce slot usage on the cluster. The improvement is at least 15% for higher reduce loads, but does not have a particular pattern. The map part of the A/B test is doing most of the heavy data pre-processing for each MR job, and the reduce part of a MR job receives a much smaller portion of the data. Thus, it is more difficult to spot the exact optimization pattern.

5. LESSONS LEARNED

In this paper, we report on optimization strategies which can be applied to a broad class of analytical tasks on Hadoop. We developed those strategies based on our experience of migrating large-scale analytical tasks (e.g. A/B testing) from a traditional data-warehousing infrastructure, like Teradata + SAS to an open-source Hadoop. Our optimization strategies benefit from exploring data dependencies within the analytical jobs, together with the probabilistic model of Hadoop

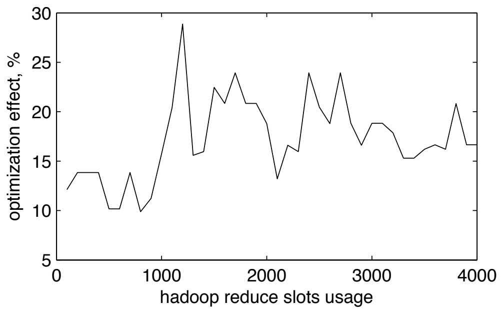


Figure 19: Optimization effect for a modified A/B test schema as a function of total Hadoop reduce slots usage

cluster load. The effectiveness of our methods for a group of independent MapReduce tasks requires further investigation.

In our experiments, we implemented MapReduce jobs using Apache Hive. However, the obtained results are portable to any other implementation language.

In order to apply our strategies, a user must have enough privileges to perform Hadoop load monitoring to calculate performance coefficients for **Equation 7** and the probabilistic Hadoop cluster load. As an option, those results can be optioned by the system administrator, and provided upon request. However, Hadoop load time series are crucial to derive the optimization strategies.

Our results are obtained for a Hadoop cluster using capacity scheduler. Optimization results provided in this paper may not be 1-to-1 applicable to those clusters running different scheduling mechanisms. While the performance **Equation 7** would remain valid, the validity of the **Algorithm 1** is yet to be confirmed for other schedulers.

6. ACKNOWLEDGMENTS

We thank Kiran Patlolla and Jesse Bridgewater from eBay Inc, Christos Faloutsos from Carnegie Mellon University, and Martijn de Jongh from the University of Pittsburgh for valuable feedback on this work. We thank Tony Thrall, manager of the EDA team at eBay Inc. for permission to publish the results of this study.

7. REFERENCES

[1] http://aws.amazon.com/elasticmapreduce/.
[2] http://developer.yahoo.com/ blogs/hadoop/posts/2011/02/capacity-scheduler/.
[3] http://pig.apache.org.
[4] http://wiki.apache.org/hadoop/.
[5] http://wiki.apache.org/hadoop/HowManyMapsAndReduces.
[6] http://www.cascading.org.
[7] http://www.ibm.com/software/analytics/spss.
[8] http://www.microstrategy.com.
[9] http://www.sas.com.
[10] http://www.scala-lang.org.
[11] http://www.tableausoftware.com.
[12] http://www.teradata.com.
[13] http://www.vertica.com.
[14] Teradata purpose-built platform pricing, http://www.teradata.com/brochures/teradata-purpose-built-platform-pricing-eb5496/.
[15] E. Baldeschwieler. Hadoop @ yahoo! - internet scale data processing. In *Cloud Computing Expo*, Santa Clara, CA, USA, Nov 2009.
[16] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
[17] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Commun. ACM*, 35(6):85–98, June 1992.
[18] D. J. DeWitt, J. F. Naughton, D. A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *Proceedings of the 18th International Conference on Very Large Data Bases*, VLDB '92, pages 27–40, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.
[19] H. Herodotou. Hadoop performance models. Technical Report CS-2011-05, Computer Science Department, Duke University, June 2011.
[20] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *PVLDB*, 4(11):1111–1122, 2011.
[21] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, pages 261–272, 2011.
[22] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data Min. Knowl. Discov.*, 18(1):140–181, Feb 2009.
[23] D. Laney. 3d data management: Controlling data volume, velocity, and variety. Technical report, Meta Group, 2001.
[24] J. Lin and A. Kolcz. Large-scale machine learning at twitter. In *SIGMOD '12 Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 793–804, New York, NY, USA, 2012. ACM.
[25] N. Parikh. Mining large-scale temporal dynamics with hadoop. In *Hadoop Summit*, San Jose, CA, Jun 20 2012.
[26] R. Pfeffer. *Teradata RDBMS*. NCR, Teradata Division.
[27] B. T. Rao and L. S. S. Reddy. Survey on improved scheduling in hadoop mapreduce in cloud environments. *CoRR*, abs/1207.0780, 2012.
[28] M. Sabah. Hadoop and cloud and netflix: Taming the social data. In *Hadoop Summit*, San Jose, CA, June 13-14 2012.
[29] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
[30] M. Stonebraker. The case for shared nothing. *IEEE Database Eng. Bull.*, 9(1):4–9, 1986.
[31] J. Tan, X. Meng, and L. Zhang. Delay tails in mapreduce scheduling. In *SIGMETRICS '12 Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pages 5–16.

[32] J. Tan, X. Meng, and L. Zhang. Delay tails in mapreduce scheduling delay tails in mapreduce scheduling. In *SIGMETRICS '12 Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pages 5–16. ACM, 2012.

[33] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer. Overlapping experiment infrastructure: Overlapping experiment infrastructure: More, better, faster experimentation. In *Proceedings 16th Conference on Knowledge Discovery and Data Mining,*, pages 17–26, Washington, DC, USA, 2010. ACM.

[34] Teradata. *Introduction to Teradata® RDBMS*. B035-1091-122A. NCR Corporation, Dec 2002.

[35] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, pages 996–1005, 2010.

[36] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 155–162, Washington, DC, USA, 2011. IEEE Computer Society.

[37] A. Verma, L. Cherkasova, and R. Campbell. Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 11–18, 2012.

[38] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: automatic resource inference and allocation for mapreduce environments. In *ICAC '11 Proceedings of the 8th ACM international conference on Autonomic computing*, pages 235–244, New York, NY, USA, 2011. ACM.

[39] A. Verma, L. Cherkasova, and R. H. Campbell. Slo-driven right-sizing and resource provisioning of mapreduce jobs. In *Workshop on Large Scale Distributed Systems and Middleware (LADIS) in conjunction with VLDB*, Seattle, Washington, 09/2011 2011.

[40] G. Wang, A. Butt, P. Pandey, and K. Gupta. A simulation approach to evaluating design decisions in mapreduce setups. In *MASCOTS*, pages 1–11, 2009.

[41] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2nd edition, Sep 2010.

[42] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin. Flex: A slot allocation scheduling optimizer for mapreduce workloads. In I. Gupta and C. Mascolo, editors, *Middleware 2010*, LNCS 6452, pages 1–20, 2010.

[43] X. Yang and J. Sun. An analytical performance model of mapreduce. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pages 306–310, 2011.

[44] M. Yong, N. Garegrat, and M. Shiwali. Towards a resource aware scheduler in hadoop. In *ICWS*, 2009.

[45] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCBEECS200955, EECS Department University of California Berkeley, 2009.

[46] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys '10 Proceedings of the 5th European conference on Computer systems*, pages 265–278, New York, NY, USA, 2010. ACM.

[47] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Automated profiling and resource management of pig programs for meeting service level objectives. In *Proceedings of the 9th international conference on Autonomic computing*, pages 53–62, New York, NY, USA, Sept. 14-18 2012. ACM.

[48] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Optimizing completion time and resource provisioning of pig programs. In *CCGRID '12 Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 811–816, Washington, DC, USA, 2012. IEEE Computer Society.