MapReduce Programming and Cost-based Optimization? Crossing this Chasm with Starfish

Herodotos Herodotou
Duke University
hero@cs.duke.edu

Fei Dong
Duke University
dongfei@cs.duke.edu

Shivnath Babu*
Duke University
shivnath@cs.duke.edu

ABSTRACT

MapReduce has emerged as a viable competitor to database systems in big data analytics. MapReduce programs are being written for a wide variety of application domains including business data processing, text analysis, natural language processing, Web graph and social network analysis, and computational science. However, MapReduce systems lack a feature that has been key to the historical success of database systems, namely, cost-based optimization. A major challenge here is that, to the MapReduce system, a program consists of black-box map and reduce functions written in some programming language like C++, Java, Python, or Ruby.

Starfish is a self-tuning system for big data analytics that includes, to our knowledge, the first *Cost-based Optimizer* for simple to arbitrarily complex MapReduce programs. Starfish also includes a *Profiler* to collect detailed statistical information from unmodified MapReduce programs, and a *What-if Engine* for fine-grained cost estimation. This demonstration will present the profiling, what-if analysis, and cost-based optimization of MapReduce programs in Starfish. We will show how (nonexpert) users can employ the *Starfish Visualizer* to (a) get a deep understanding of a MapReduce program's behavior during execution, (b) ask hypothetical questions on how the program's behavior will change when parameter settings, cluster resources, or input data properties change, and (c) ultimately optimize the program.

1. INTRODUCTION

MapReduce is a relatively young framework—both a programming model and an associated run-time system—for large-scale data processing [4]. *Hadoop* [5] is a popular open-source implementation of MapReduce that many academic, government, and industrial organizations use in production deployments. Hadoop is used for applications such as Web indexing, data mining, report generation, log file analysis, machine learning, financial analysis, scientific simulation, and bioinformatics research. Cloud platforms make MapReduce an attractive proposition for small organizations that need to process large datasets, but lack the computing and human resources of a Google or Yahoo! to throw at the problem. *Elastic MapReduce*, for example, is a hosted platform on the Amazon cloud where users can instantly provision Hadoop clusters to perform data-intensive tasks; paying only for the resources used.

A MapReduce program p is run on input data d and cluster resources r as a *MapReduce job* $j = \langle p, d, r, c\rangle$. c represents a set of *configuration parameter settings* needed in order to fully specify how the job should execute on the cluster. Choices for settings in c include (but are not limited to):

1. Degree of parallelism. The execution of j consists of running parallel map and reduce tasks. These tasks may run in multiple *waves* depending on the number of execution slots in r.
2. The amount of memory to allocate to each map (reduce) task of j to buffer its outputs (inputs).
3. The settings for the multiphase external sorting used to group map-output values by key.
4. Whether the output data from the map (reduce) tasks should be compressed before being written to disk.
5. Whether a given combine function should be used to preaggregate map outputs before their transfer to reduce tasks.

Hadoop has more than 190 configuration parameters out of which 10-20 parameters can have significant impact on job performance. Today, the burden falls on the user who submits the MapReduce job to specify settings for all configuration parameters. For any parameter whose value is not specified explicitly during job submission, default values—either shipped with the system or specified by the system administrator—are used. Higher-level languages for MapReduce like HiveQL and Pig Latin have developed their own *hinting* syntax for setting parameters.

The impact of various parameters as well as their best settings vary depending on the MapReduce program, input data, and cluster resource properties. Personal communication, our own experience [1, 7], and plenty of anecdotal evidence on the Web indicate that finding good configuration settings for MapReduce jobs is time consuming and requires extensive knowledge of system internals.

To automate this process, we developed Starfish [7], a self-tuning system for big data analytics. Starfish, which is built on Hadoop, aims to enable Hadoop users and applications to get good performance automatically without any need on their part to understand and manipulate the many tuning knobs available. Starfish includes a *Cost-based Optimizer* to find good configuration settings automatically for arbitrary MapReduce programs. The Optimizer requires the use of two other components: a *Profiler* that instruments unmodified MapReduce programs dynamically to generate concise statistical summaries of MapReduce job execution; and a *What-if Engine* to reason about the impact of parameter configuration settings, as well as data and cluster resource properties, on the performance of MapReduce jobs.

*Supported by NSF grants 0644106 and 0964560

In this demonstration, we will present the uses and contributions of each component in optimizing MapReduce program execution:

- The information collected by the Profiler helps in understanding the job behavior as well as in diagnosing bottlenecks during job execution.
- The What-if Engine can predict the performance of a MapReduce job j, allowing the user to study the effects of configuration parameters, cluster resources, and input data on the performance of j; without actually running j.
- The Cost-based Optimizer can find the optimal configuration settings for j, and also help understand why the current settings are possibly suboptimal.

2. COST-BASED OPTIMIZATION

Consider a MapReduce job $j = \langle p, d, r, c \rangle$ that runs program p on input data d and cluster resources r using configuration parameter settings c. Job j's performance can be represented as:

$$perf = F(p, d, r, c) \quad (1)$$

Here, *perf* is some performance metric (e.g., execution time) of interest for jobs that is captured by the *cost model* F. Optimizing the performance of program p for given input data d and cluster resources r requires finding the configuration parameter settings c_{opt} that give the optimal value of *perf*. The What-if Engine uses a set of analytical and simulation models to implement the cost function F, whereas the Profiler gathers the necessary information for performing cost-based optimization. The three components of our solution are described next.

2.1 Profiler

The Profiler is responsible for collecting *job profiles*: statistical summaries of MapReduce job execution. A job profile is a vector of fields where each field captures some unique aspect of data-flow or cost estimates during job execution. Data-flow estimates represent information regarding the amount of bytes and key-value pairs processed during the job's execution, as well as key-value distributions for input, intermediate, and output data. Cost estimates represent execution time and resource usage, including the usage trends of CPU, memory, I/O, and network resources during the job's execution. The full listing of profile fields can be found in [6].

The information included in a job profile is at the fine granularity of phases within the map and reduce tasks of a MapReduce job execution. This feature is crucial to the accuracy of decisions made by the What-if Engine and the Cost-based Optimizer. Apart from using job profiles in answering what-if questions, the job profiles also help in understanding the job behavior as well as in diagnosing bottlenecks during job execution.

The Profiler uses *dynamic instrumentation* [3] to collect run-time monitoring information from unmodified MapReduce programs running on Hadoop. Dynamic instrumentation is now a popular technique to understand, debug, and optimize complex systems . Our current implementation of the Profiler uses the *BTrace* dynamic instrumentation tool for Java [2]. When Hadoop runs a MapReduce job, the Profiler dynamically instruments selected Java classes internal to Hadoop to collect raw monitoring data. The raw data will undergo a series of post-processing steps in order to construct a concise job profile. First, the raw data collected for each profiled map or reduce task is processed to give fields in a *task profile*. The map (reduce) task profiles are further processed to give representative map (reduce) task profiles. The task profiles also contain all individual key-value flows, which are used to compute the key-value distributions across all map and reduce tasks.

Three features are noteworthy. First, the Profiler only instruments the MapReduce framework, and not user-written programs. Hence, profiling works irrespective of whether the user submits the MapReduce program in Java, in Python/Ruby using *Hadoop Streaming*, or in C++ using *Hadoop Pipes*. Second, dynamic instrumentation can be turned on or off seamlessly at run-time, incurring zero overhead when turned off; an appealing property in production deployments. Third, task-level sampling can be used to generate approximate job profiles quickly.

2.2 What-if Engine

The What-if Engine is given four inputs when asked to predict the performance of a MapReduce job j:

1. Job profile generated for j by the Profiler.
2. New configuration settings c to run j with.
3. Size, layout, and compression information of the input dataset d on which j will be run. Note that this input dataset can be different from the dataset used while generating the job profile.
4. Cluster setup and resource allocation r that will be used to run j. This information includes the number of nodes and network topology of the cluster, the number of map and reduce task slots per node, and the memory available for each task execution.

These inputs and a detailed set of analytical and simulation models we developed are used to generate a *virtual job profile* for the hypothetical job j' that represents j run with the new configuration settings. The virtual job profile is used by a *Task Scheduler Simulator*, along with the models and information on the cluster resources r, to simulate the scheduling and execution of the map and reduce tasks of j'; from which the desired performance metrics are estimated. The full details are given in [6].

2.3 Cost-based Optimizer

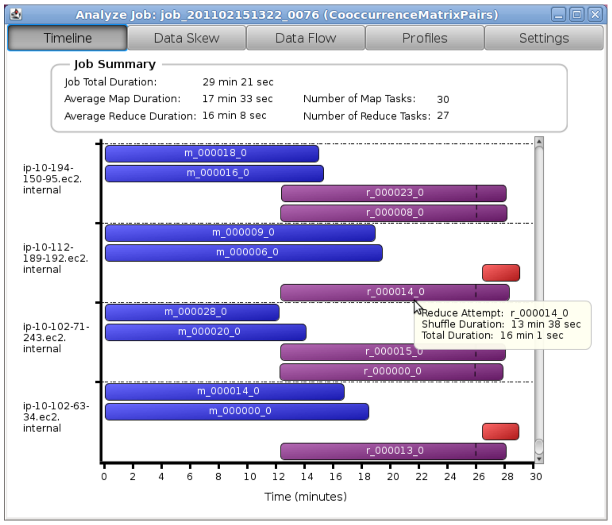
Given a MapReduce program p to be run on input data d and cluster resources r, the Optimizer must find the setting of configuration parameters $c_{opt} = \underset{c \in S}{\operatorname{argmin}} F(p, d, r, c)$ for the cost model F represented by the What-if Engine over the space S of configuration parameter settings. The Optimizer addresses this problem by making what-if calls with settings c of the configuration parameters selected through an enumeration and search over S. In order to provide both efficiency and effectiveness, the Optimizer must minimize the number of what-if calls while finding near-optimal configurations.

For this purpose, the Optimizer uses *Recursive Random Search (RRS)*, a recent technique developed to solve black-box optimization problems [9]. RRS first samples the space S randomly to identify promising regions that contain the optimal setting with high probability. These regions are then sampled recursively, and the regions either move or shrink gradually to locally-optimal settings based on the samples collected. RRS then restarts random sampling to find a more promising region to repeat the recursive search.

3. DEMONSTRATION PLAN

The demonstration will use the *Starfish Visualizer*, a new graphical user interface that we have developed. Users employ the Visualizer to (a) get a deep understanding of a MapReduce job's behavior during execution, (b) ask hypothetical questions on how the job behavior will change when parameter settings, cluster resources, or input data properties change, and (c) ultimately optimize the job. Hence, we categorize the core functionalities of the Visualizer into *Job Analysis*, *What-if Analysis*, and *Job Optimization*. For each functionality, the Visualizer offers five different views:

1. *Timeline views*, used to visualize the progress of job execution at the task level.
2. *Data-skew views*, used to identify the presence of data skew in the input and output data for map and reduce.

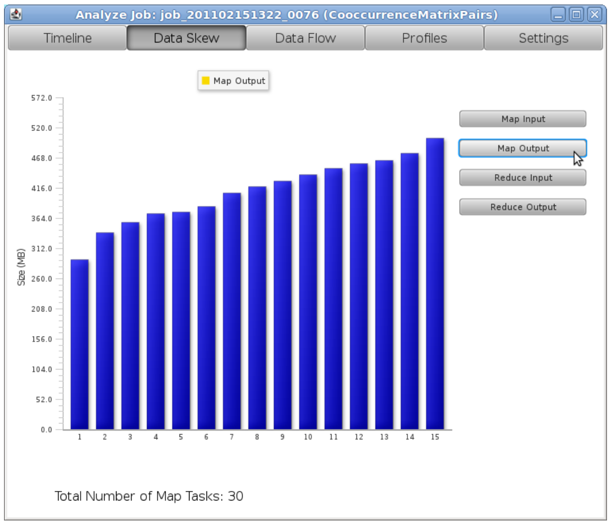


Job Summary

Job Total Duration:	29 min 21 sec		
Average Map Duration:	17 min 33 sec	Number of Map Tasks:	30
Average Reduce Duration:	16 min 8 sec	Number of Reduce Tasks:	27

Reduce Attempt: r_000014_0
Shuffle Duration: 13 min 38 sec
Total Duration: 16 min 1 sec

Figure 1: Execution timeline of the map and reduce tasks of a MapReduce job in a Hadoop cluster running on Amazon EC2



Total Number of Map Tasks: 30

Figure 2: A histogram showing the data-skew of the map output produced by a MapReduce job

3. *Data-flow views*, used to visualize the flow of data among the nodes of a Hadoop cluster, and between the map and reduce tasks of a job.
4. *Profile views*, used to show the detailed information exposed by the job profiles, including the phase timings within the tasks.
5. *Settings views*, used to list the configuration parameter settings, cluster setup, and the input data properties during job execution.

We will demonstrate the Visualizer's functionalities in order, and show how the user can obtain deep insights into a job's performance from each view in each case. The jobs considered will represent popular MapReduce programs used in different domains: text analytics (*WordCount*), natural language processing (*Word Co-occurrence*), analysis of large hyperlink graphs (*LinkGraph*, *PageRank*), and business data processing (*Join*, *TeraSort*) [8].

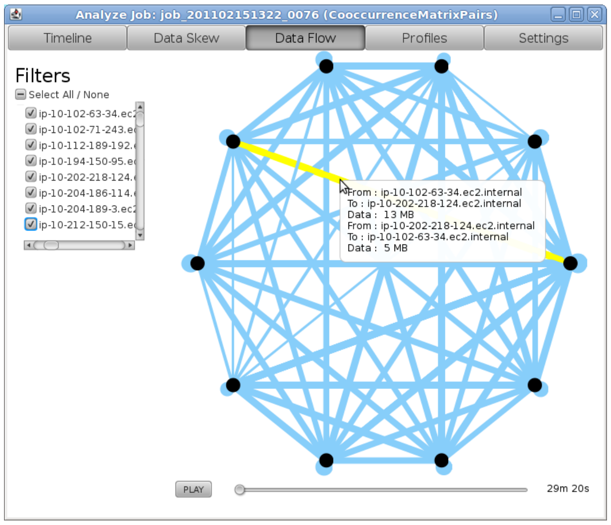
3.1 Job Analysis

When a MapReduce job executes on a Hadoop cluster, the Profiler collects a wealth of information including logs, counters, resource utilization metrics, and profiling data. Figure 1[1] shows the execution timeline of map and reduce tasks that ran during a MapReduce job execution. The user can get information such as how many tasks were running at any point in time on each node, when each task started and ended, or how many map or reduce *waves* occurred during job execution. The user is able to quickly spot any high variance in the task execution times, and discover potential load-balancing issues. Moreover, Timeline views can be used to compare different executions of the same job run at different times or with different parameter settings. Comparison of timelines shows whether the job behavior changed over time, as well as helps understand the impact of changing parameter settings on job execution.

While the Timeline views are useful in identifying computational skew, the Data-skew views (shown in Figure 2) can readily help identify the presence of skew in the data consumed and produced by the map and reduce tasks. Data skew in the reduce tasks usually indicates a strong need for a better partitioner in a MapReduce job. Data skew in the map tasks corresponds to properties of the input data, and may indicate the need for a better partitioner in the producer job that generates the input data.

The Data-skew views are complemented by the Data-flow views used to identify data skew across the Hadoop nodes. Figure 3

[1]All figures are actual screenshots from the Starfish Visualizer.



From : ip-10-102-63-34.ec2.internal
To : ip-10-202-218-124.ec2.internal
Data : 13 MB
From : ip-10-202-218-124.ec2.internal
To : ip-10-102-63-34.ec2.internal
Data : 5 MB

Figure 3: Visual representation of the data-flow among the Hadoop nodes during a MapReduce job execution

presents the data flow among the nodes during the execution of a MapReduce job. The thickness of each line is proportional to the amount of data that was shuffled between the corresponding nodes. The user also has the ability to specify a set of filter conditions (see the left side of Figure 3) that allows her to zoom in on a subset of nodes or on the large data transfers. An important feature of the Visualizer is the *Video mode* that allows users to play back a job execution from the past. Using the Video mode, the user can inspect how data was processed and transfered between the map and reduce tasks of the job, and among nodes and racks of the cluster, as time went by.

The Profile views help visualize the job profiles, namely, the information exposed by the profile fields at the fine granularity of phases within the map and reduce tasks of a job; allowing for an in-depth analysis of the task behavior during execution. For example, Figure 4 displays the breakdown of time spent on average in each map and reduce task. The Profile views also form an excellent way of diagnosing bottlenecks during task execution. It stands out in Figure 4 that the time spent merging the map output data at the reduce tasks contributes the most to the total execution time; in-

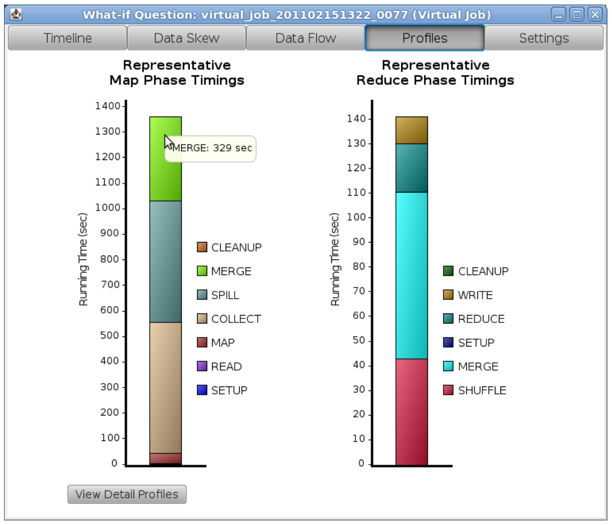
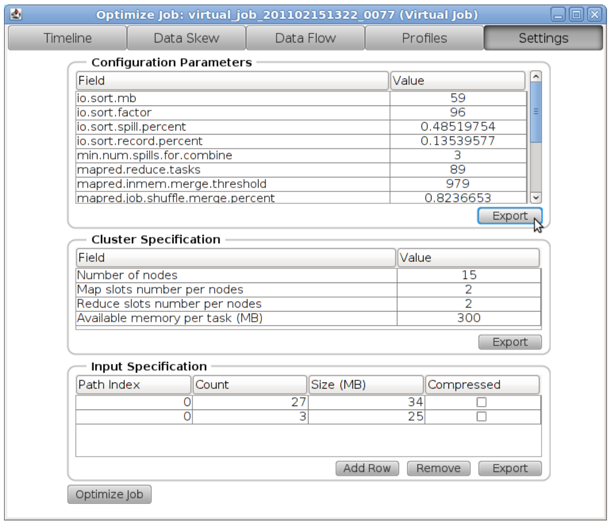


Figure 4: Map and reduce time breakdown from the virtual profile for a MapReduce job



Configuration Parameters

Field	Value
io.sort.mb	59
io.sort.factor	96
io.sort.spill.percent	0.48519754
io.sort.record.percent	0.13539577
min.num.spills.for.combine	3
mapred.reduce.tasks	89
mapred.inmem.merge.threshold	979
mapred.job.shuffle.merge.percent	0.8236653

Cluster Specification

Field	Value
Number of nodes	15
Map slots number per nodes	2
Reduce slots number per nodes	2
Available memory per task (MB)	300

Input Specification

Path Index	Count	Size (MB)	Compressed
0	27	34	☐
0	3	25	☐

Figure 5: The optimal configuration settings found by the Optimizer, as well as the cluster and input data specifications

dicating that the corresponding configuration parameters have settings that are potentially suboptimal.

Finally, the Settings view (see Figure 5) lists the most important Hadoop configuration parameters used during the execution of a MapReduce job, as well as the cluster setup and input data properties. The cluster setup is summarized as the number of nodes, the average number of map and reduce slots per node, and the memory available for each task execution. The input data properties include the size and compression of each input split processed by a single map task. The user also has the option of exporting any of the above settings in XML format.

3.2 What-if Analysis

The second core functionality provided by the Starfish Visualizer is the ability to answer hypothetical questions about the behavior of a MapReduce job when run under different settings. This functionality allows users to study and understand the impact of configuration parameter settings, cluster setup, and input data properties on the performance of a MapReduce job.

For instance, the user can ask a what-if question of the form: "How will the execution time of a job change if the number of reduce tasks is doubled?" The user can then use the Timeline view to visualize what the execution of the job will look like under the new settings, and compare it to the current job execution. By varying the number of reducers (or any other configuration parameter), the user can determine the impact of changing that parameter on the job execution. Under the hood, the Visualizer invokes the What-if Engine to generate a virtual job profile for the job in the hypothetical setting (recall Section 2.2).

Furthermore, the user can investigate the behavior of MapReduce jobs when changing the cluster setup or the input specification. This functionality is useful in two scenarios. First, many organizations run the same MapReduce programs over multiple datasets with similar data distribution, but different sizes. For example, the same report-generation MapReduce program may be used to generate daily, weekly, and monthly reports. Or, the daily log data collected and processed may be larger for a weekday than the data for the weekend. By modifying the input specification, the user can ask what-if questions on the job behavior when the job is run using datasets of different sizes.

Another common use-case is the presence of a development/test cluster for generating job profiles. In many companies, developers use a small test cluster for testing and debugging MapReduce programs over small (representative) datasets before running the programs, possibly multiple times, on the production cluster. Again, the user can modify the cluster setup in order to determine in advance how the jobs will behave on the production cluster. These novel capabilities are immensely useful in Hadoop deployments; a detailed evaluation is provided in [6].

3.3 Job Optimization

Perhaps the most important functionality of the Visualizer comes from how it can use the Cost-based Optimizer to find good configuration settings for executing a MapReduce job on a Hadoop cluster. The user can then export the configuration settings as an XML file that is used when the same program has to be run in future. At the same time, the user can examine the behavior of the optimal job through the other views provided by the Visualizer.

Similar to the What-if Analysis functionality, the user can modify the cluster and input specifications before optimizing a MapReduce job. Hence, the user can obtain good configuration settings for the same MapReduce program executed over different input datasets and different clusters (per the two usage scenarios presented above). In addition, the user can take advantage of the sampling capabilities of the Profiler to quickly collect a job profile on a sample of the input data. The user can then modify the input specifications and find the optimal settings to use when executing the MapReduce program over the full (or a different) dataset.

4. REFERENCES

[1] S. Babu. Towards Automatic Optimization of MapReduce Programs. In *SOCC*, 2010.

[2] *A Dynamic Instrumentation Tool for Java*. kenai.com/projects/btrace.

[3] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic Instrumentation of Production Systems. In *USENIX*, 2004.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[5] *Apache Hadoop*. http://hadoop.apache.org/.

[6] H. Herodotou and S. Babu. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. *PVLDB*, 4, 2011.

[7] 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*, 2011.

[8] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool, 2010.

[9] T. Ye and S. Kalyanaraman. A Recursive Random Search Algorithm for Large-scale Network Parameter Configuration. *SIGMETRICS*, 2003.