Abstract

This interim report aims at providing detailed background information and practical methodology of our final project “Dynamic Spark Optimizer”, along with the latest progress and milestones of future work. This project focuses on implementing an automatic optimizer for the cluster computing platform, Apache Spark, aiming at approaching optimal performance for various job types. The project is accomplished by Greg Liao Ziheng and Eva Zhou Dan, supervised by Professor Francis Lau from The Computer Science Department of HKU. Greg Liao mainly takes charge of the development of infrastructure while Eva Zhou concentrates more on the establishment of analytic models and optimizing algorithms.

Acknowledgement

We would like to express our gratitude to Prof. F.C.M. Lau and Mr. Liang, for their guidance and their support on this project. Also we would like to thank Dr. Wu for taking her time examining this project, and Dr. Chenggang Zhang and Lenovo for generously providing the testing environment.
# Table of Contents

ABSTRACT .................................................. 2

ACKNOWLEDGEMENT .................................... 2

BACKGROUND ............................................ 4

RELATED WORKS ........................................ 6

OBJECTIVE & DELIVERABLES ....................... 7

ENGINEERING JUSTIFICATIONS .................... 7

EXPERIMENTAL ENVIRONMENT ..................... 8

METHODOLOGY .......................................... 9

CURRENT PROGRESS AND FUTURE WORK ........ 13

OBSTACLES ENCOUNTERED ........................... 14

APPENDIX I – ABBREVIATIONS ...................... 16

APPENDIX II – LIST OF FIGURES AND TABLES .... 16

APPENDIX III – REFERENCES ........................ 16
Background

The technology of cluster computing has been developing rapidly since Google introduced MapReduce in 2003 when a single server could no longer handle the massive computation brought by numerous data generated every day. Big corporations started to use clusters of servers to raise their competitiveness in productivity and efficiency, and that made room for the booming industry of cluster computing platforms.

Apache Spark, defined as “an open source cluster computing platform designed to be fast and for general purposes” [1], was created by AMPLab of UC Berkeley just 2 years ago and aimed at overcoming limitations of the state-of-the-art cluster computing framework MapReduce. Spark is usually considered to have achieved its designed goals, which are “fast” and “for general purpose”, and as a matter of fact, it is now used as the major cluster computing platform in many big companies. On the fast side, Spark is the first mature cluster computing framework that utilizes memory to boost the performance, instead of relying largely on hard disks. As claimed by its inventors, it achieved over 100X performance gain over its predecessor Hadoop MapReduce on tasks processed in memory and over 10X in those running on hard disks. On the general-purpose side, Spark can work seamlessly on extant cluster management platforms with a support for almost all major data processing job types handled by clusters (Figure 1[2] below shows the abstract hierarchy and core functions of cluster computing system). The high compatibility with cluster management platforms and the outstanding performance on speed makes Spark a wise choice both technologically and commercially, which may explain why it becomes the popular hit in such a short time.
Although Spark holds some invincible strengths, it also has limitations. One fact about Spark is that its parameter set is quite huge and complicated, making manually tuning/optimizing Spark (mostly via tuning of parameters) a highly time consuming task. Also, due to the fact that different clusters may serve for disparate purposes, their hardware configurations and workload natures often vary a lot, which will lead to huge distinctions between different cluster/job’s optimal configurations. As a result, in most situations a cluster needs to be continuously tuned before it could approach its maximum effectiveness. However, manually optimizing a spark cluster is costly in both human resources and time. Thus an entity using any sort of cluster computing platform naturally thirsts for a tool to optimize its performance automatically, otherwise it may suffer from low efficiency. The idea of creating an automatic and dynamic optimizer of spark occurred after an investigation of the industry, in which we discovered that there was no similar product on the market. Nowadays, scale of data processed on cluster computing platforms usually goes up to multi-TBs (1 TB = 1024 GB = 1,048,576 MB), PBs (1 PB = 1024 TB) or even larger. Under such circumstances, optimization on the platform becomes so important that even a very small improvement on cluster efficiency could bring along a huge boost on productivity.

In the remaining part, we will firstly review the related research in this field, and secondly
state the incoming deliverables, engineering justifications and the experimental scope of our project, then elaborate the detailed methodology, and finally discuss the difficulties encountered and clarify the current progress of the project. Schedule of work and milestones will be attached in the section *Current Progress and Future Work.*

**Related Works**

Similar studies of automated tuning were conducted on other cluster computing platforms such as Hadoop which is the precursor of Spark. Herodotou. et al. [3] proposed a self-tuning system aiming at bringing a Hadoop cluster to its best performance. The research worked out a complete system “Starfish” which was able to tune a Hadoop cluster in 3 levels, namely the Workload-level, the Workflow-level and the Job-level. Through multi-level tuning the system can achieve a significant performance gain over untuned clusters. After studying its feasibility, we believe that similar optimization methods in terms of job-level tuning, which includes profiling, job completion time prediction and job features mining could be transplanted and applied to Spark optimizers.

Another research conducted by Delimitrou & Kozyrakis [4] provided insights on how could different incoming jobs submitted to a Spark cluster be classified. In the cluster management system “QUASAR” which was proposed in this study, they tested incoming jobs based on their performing patterns over scaling up, scaling out, heterogeneity and interference factors. Through profiling on a small amount of cluster resources, these tests successfully classified incoming jobs in a time-efficient way. Results suggested that such testing method is a reliable way to classify jobs, with pretty high classification accuracy and relatively low overhead.

A tool called SparkBench developed by Li et al. at 2015 [5] provided a viable method of conducting profiling tests, as well as testing performance gains on clusters with the optimizer. As its name suggests, SparkBench aims to provide a benchmark for quantitatively measuring the performance of a Spark cluster, which could be crucial to the development and testing of our optimizer. The benchmark suite also has over 10 different types of jobs which features distinct nature in their resource consumption
patterns, i.e. some of the jobs are CPU-intensive while others are memory intensive. As SparkBench is generally recognized as a scientific benchmark for Spark which could objectively reflect the cluster’s performance, our project could use it as the major performance test engine.

**Objective & Deliverables**

Our deliverable, a dynamic optimizer which integrates itself with Spark cluster, aims at satisfying the needs for a better and more intelligent performance of Apache Spark. The optimizer will be able to accept any incoming spark job, classify it into one of the preset categories, use the category’s specific configurations to run the job, as well as gathering information while the job is running to continuously optimize the job’s performance. After the job is completed, the optimizer will use the information collected during execution to improve existing models. The optimizer will most likely be a command line tool without an GUI. Along with the optimizer a paper will be delivered to explain the details of the optimizer’s methodology as well as related test results.

**Engineering Justifications**

The first important justifications occurred was picking Apache Spark as the target cluster computing framework. Among various cluster computing frameworks, Apache Spark is distinguished for its high efficiency and broad applicability. Compared to its counterparts, such as Hadoop, Spark is faster since it directly reads data and does computation from memory rather than files in and out from hard disks. Furthermore, not only can Spark be set up alone, it also can be established across different platforms like Yarn and Mesos, which gives Spark a huge advantage over other cluster computing frameworks and makes it one of the most advanced frameworks in this field.

Java then, were chosen to be the major implementing programming language for the project. Java is commonly acknowledged as a “fast” programming language based on
several reasons. First, Memory allocation and de-allocation in java are fast and cheap. Second, Object instantiation and object-oriented features are extremely fast to use (even faster than C++ in some cases). Third, synchronization and multi-threading are easy to use and rather efficient. The advantage of multi-threading is commonly acknowledged as it basically brings along an extra 100% to 300% speed boost vs. standard, single-threaded code. Moreover, multi-threading provides improved stability in the real testing on clusters due to its greater ability on tolerating systematic errors and thus reduce the possibility of system crashes. Based on above reasons, Java stands out to be chosen as the major programming language that we will use in this project due to its superior speed and stability.

**Experimental Environment**

At the first stage of testing, we used idle production clusters of Lenovo as the testing environment, however due to the concern of commercial confidentiality, Lenovo stopped providing collaboration in the November, 2016. Then we transferred to Amazon EC2 to build a new experimental environment. Specifications of the two testing environments are listed below.

A. Original Lenovo testing environment:
   a. Hardware configurations:
      i. Testing cluster: 8* Virtual Machines
      ii. Processor: Intel E5-2650V3@2.4GHz * 8 cores, 1 core per VM
      iii. Memory: 15GB per VM
      iv. Hard Drive: 3TB per VM
      v. Network: 1000Mbps network
   b. Software configurations:
      i. Operating System: CentOS 7.0
      ii. Spark version: Spark 1.6.1
      iii. Java version: Oracle JDK 8
   c. The testing servers were accessed by remote connection with SSH through
VPN.

B. Current Amazon EC2 testing environment:
   a. Hardware configurations:
      i. Testing cluster: 4-8 Amazon EC2 i2.xlarge instances
      ii. Processor: 4x Intel Xeon E5-2670v2 vCPU (per instance)
      iii. Memory: 30GB Memory (per instance)
      iv. Hard Drive: 800GB SSD (per instance)
      v. Network: 1000 Mbps Network
   b. Software configurations:
      i. Operating System: CentOS 6.7 (64 bit)
      ii. Spark version: Spark 1.6.2 / Spark 2.1.0
      iii. Java version: Oracle JDK 8
      iv. Benchmark: SparkBench
   c. The testing servers are accessed by remote connection with SSH.

**Methodology**

The dynamic spark optimizer will be implemented in three parts. The first part, the classifier, will be able to profile an incoming job and classify incoming spark jobs into different categories based on profiling results. For each category there will be a set of configurations which are specifically optimized regarding their specific characteristics to achieve better performance. The job will then be executed by the executor, which will gather real-time information of the running job and detect & handle the abnormalities. When the job is finished, the analyzer will run data mining task on the job’s performance data (gathered by cluster monitoring tools) to further enhance statistical models used in classification and parameter optimization. These three parts will be further explained in the next paragraphs. The relationship between these components are indicated in Figure 2 below.
Not as easy as it may sound to be, classifying a spark job could face a number of problems. These problems must be answered before we could be able to build an effective optimizer for Spark. For example, the standards of classification may vary, as a job could be classified by different criteria. As a result, to ensure the effectiveness of our optimizer, we need to take into account the most relevant information gathered from profiling which could guide the parameter tuning process. For example, timings view which suggests execution time distribution among stages, data-flow view which gives the amount of data processed during various stages, gc view which reveals garbage collection frequency and time and resource-level view which shows resource consumption patterns. Since these features of a job are highly related to the job’s overall performance and can reveal critical information such as memory consumption and garbage collection statistics which is sensitive to Spark parameter changes, classification results based on these features should be representative enough to make further tuning on each category effective. Another problem is that different jobs which are classified into the same category can still be different in some ways, thus an optimization to the category may not necessarily result in near-optimal performance for all the jobs which are in that category. After discussion
with Prof. Francis and Dr. Wu during the 1st presentation, we came up with an alternative solution, that is to design an algorithm which could apply to the various views gathered during profiling, and instead of doing a clustering to the jobs, we use the algorithm to determine the near-optimal parameter set for each of individual jobs. We tried to examine our proposed solutions to these problems by testing over various types of Spark jobs on our testing cluster. Following the methods applied in QUASAR and Starfish, we tested jobs to gather critical information for building the classifier. We used SparkBench as a testing tool and utilized Spark history server as well as profiling tools like Ganglia to capture the clusters’ resource consumption information and the job’s execution statistics. By further studying the testing results we are now building a valid classifier with a highly accurate and effective classification algorithm. The categories in this classifier are supposed to be sensitive to configuration changes and representative in terms of features like resource consumption and garbage collection. When a new job is submitted to the classifier, part of the job (usually under 20%) will be profiled by the same tests to measure its performance, and the result will be categorized by the classification algorithm that we developed.

When a job gets classified into one of the categories by the classifier, it will then execute using the category’s specific configurations for Spark. The configurations will be generated by exhaustively testing different configuration sets and selecting the near-optimal one. Alternatively, if we managed to develop an algorithm to determine each task’s specific optimized parameter set, we just apply that algorithm to get an optimized set of runtime configuration. While the job is running, profiling by means of dynamic instrumentation will be applied, gathering information about the current job. If during the execution any abnormality is discovered by the profiler, the executor will act accordingly to enhance the performance, including shutting down problematic spark executors, migrating stragglers, etc. Once the job is finished, the information gathered by the profiler will proceed to the analyzer which, after performing data mining tasks, will update the existing criterial in classifier as well as the configurations in the particular category. A predictor can also be implemented to predict the job’s performance (in this case, it’s the job execution time) based on the profiling results, by methods brought up by Wang and Khan[6]. The predictor could be useful in the analyzer and future improvements of this
optimizer if we would like to add resource-management level tuning features. This feature will only be implemented if there is extra time to finish it. The executor and analyzer will rely largely on the classifier as well as test data, and we will include more details of them in the final report.

Note that the three-part breakdown of the optimizer will be realized by a two-part implementation process, one is core infrastructure and the other is data processing. Either part will be implemented independently, in the form of modules of a modular system. Modules will be able to integrate with each other to form a functioning optimizer. Figure 3 illustrate the relationship between data processing modules and infrastructure modules.

Figure 3. Data analytical modules in the optimizer (light blue blocks)
## Current Progress and Future Work

<table>
<thead>
<tr>
<th>Time</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>August-September</td>
<td>✓ Background research and literature review (finished)</td>
</tr>
<tr>
<td></td>
<td>✓ Discuss with supervisor about feasibility assessment of our project (finished)</td>
</tr>
<tr>
<td>October-November</td>
<td>✓ Do pre-tests base on benchmarks and collect key information (finished)</td>
</tr>
<tr>
<td></td>
<td>✓ Do primary analysis on collected data to help understand the correlation between configuration sets and performance flaws (finished)</td>
</tr>
<tr>
<td>December</td>
<td>✓ Conduct a demo model based on standard benchmarks (in progress, delayed)</td>
</tr>
<tr>
<td></td>
<td>✓ Adjust the model based on the demo feedbacks (in progress, delayed)</td>
</tr>
<tr>
<td>January-February</td>
<td>✓ Establish real statistic models and build a prototype of optimizer (in progress)</td>
</tr>
<tr>
<td></td>
<td>✓ Apply more tests and make adjustments accordingly (in progress)</td>
</tr>
<tr>
<td>February-March</td>
<td>✓ Refine the optimizer</td>
</tr>
<tr>
<td></td>
<td>✓ Deliver a draft report</td>
</tr>
<tr>
<td>April</td>
<td>✓ Implement credibility tests</td>
</tr>
<tr>
<td></td>
<td>✓ Finish report</td>
</tr>
<tr>
<td></td>
<td>✓ Final delivery</td>
</tr>
</tbody>
</table>

Table 1. Schedule and Milestones

In the last several months, we have done sufficient work to complete the background research and literature review, mainly investigating related outcomes from published papers and following the new progress from institutions. Beyond that we were also getting educated with associative tools, such as SparkBench (a benchmark for quantitatively measuring the performance of a Spark cluster), and anaconda (an in-line interactive python IDE for data pre-processing and algorithms implementation), which could prepare us to be proficient on the project.
A fresh Spark cluster for testing was installed on the cluster environment and several trail-tests by using SparkBench were conducted to adjust the parameters to a proper state. Despite a huge data loss due to the misconfiguration of clusters, part of the initial batch of results were still retrieved and analyzed, which formed a solid ground for further development of the optimizer.

The tests mentioned above aimed to reproduce QUASAR’s scale-up and scale-out tests, as well as getting the timings, data-flow and resource-level view from these tests, as mentioned in the paper of Starfish. Examinations on the results suggested that the classification approaches used in related studies are applicable to Spark, as various types of jobs displayed distinctive resource consumption patterns. Such tests are still ongoing and detailed test results and analysis will be elaborated in the final report. Currently we are working on constructing the classifier which utilizes the test results, while more tests are running concurrently on our test servers.

In the coming months, we will concentrate on disparate work of construction. Greg will have more work on establishing the framework of the optimizer while Eva will perform quantitative analysis on data collected from tests.

## Obstacles Encountered

For the background research and literature review part, there existed some difficulties in understanding the statistical models and algorithms that were used in the studies, which caused a much longer time than expected for us to fully comprehend the ideas conveyed.

In configuring the clusters’ environments, there were a lot of bugs which greatly halted the process of software installation. Solutions of those problems were found by online search, and were marked down in documents in case the servers break down in the future and we may need to configure them again.

When we run tests by using the testing cluster, there were connectivity issues that interrupted remote connection to the server. This problem didn’t influence the functionality of the tests, but it had a negative impact on working efficiency. Several
methods have been applied to alleviate this phenomenon, however the problem still remains unresolved at the time of writing.

We suffered from data loss due to unexpected server shutdown. Once the Lenovo data centre’s air conditioner broke down and the whole testing cluster went down. After we migrated to Amazon EC2, the spot instances terminated themselves once due to improper settings. To prevent potential data loss, we tried to maintain a high-availability backup server which will backup the data generated every day by the testing cluster.
Appendix I – Abbreviations

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>Apache Spark™</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>Amazon Elastic Compute Cloud</td>
</tr>
</tbody>
</table>

Appendix II – List of Figures and Tables

<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Abstract structure and core functions of cluster computing system</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Relation between three parts of the optimizer</td>
<td>10</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Data analytical modules in the optimizer (light blue blocks)</td>
<td>12</td>
</tr>
<tr>
<td>Table 1</td>
<td>Schedule and Milestones</td>
<td>13</td>
</tr>
</tbody>
</table>

Appendix III – References