Efficient, Problem-Tailored Big Data Processing Using Framework Delegation*

Nickolas Davis  
Dept. of Computer Science & Engineering  
New Mexico Tech  
Socorro, NM, USA  
n dav01@nmt.edu

Matthew Broomfield  
Dept. of Computer Science & Engineering  
New Mexico Tech  
Socorro, NM, USA  
mbroomfi@nmt.edu

Abdelmounaam Rezgui  
Dept. of Computer Science & Engineering  
New Mexico Tech  
Socorro, NM, USA  
rezgui@cs.nmt.edu


ded the most efficient runts of technoloies and the approach to this solution.  
, this paper.  
d small problems [9] [12] dual framework in tools created to attempt to fix computes best suits.  
e usefulness of this.  

21 [45x112] [7].  

The rise of the Internet of Things, social networking, and embedded connectivity has led to an explosion of available data. In order to better analyze this big data, many different tools have been created that can process the data efficiently. However, the increase in the amount of tools available makes it more difficult to determine which one will provide the most efficient solution to a given big data problem. In this paper, we present a delegation system that takes various frameworks and problem parameters as input and computes the best framework to use for a specific big data problem. To evaluate our system, we used two big data processing frameworks, namely, Hadoop MapReduce and AJIRA, with problem size as an input parameter. Preliminary results show that the system is able to select the most optimal big data processing framework for a given problem 90% of the time. Moreover, the proposed delegation system introduces only an additional 0.01% overhead when compared to the individual framework in terms of execution time.

Keywords—Big data, Hadoop MapReduce; AJIRA.

I. INTRODUCTION

The proliferation of data generating technologies and the Internet of Things have led to a data deluge. As of 2014, every day 2.3 zettabytes (2.3×10^21) of data were created [1]. This rapid generation of data has in turn increased the demand for large scale data analysis tools. One such big data analysis tool is Hadoop MapReduce. Hadoop manages file systems to a cluster of computing machines [2]. MapReduce is a software framework that allows developers to write programs that process massive amounts of data in parallel across a distributed cluster of computers [3]. Combined, Hadoop and MapReduce create an excellent tool that allows users to analyze big data sets. However, because of its notorious overhead, there are many situations in which Hadoop MapReduce does not operate optimally. Two example domains are: iterative problems and small problems [9] [12] [7].

There have been various tools created to attempt to fix Hadoop MapReduce’s shortcomings in these areas. Example systems include: Vertica, VoltDB, Spark, Hive and Pig [11]. It is generally the goal of these frameworks to improve performance for a specific set of tasks. An example would be to improve energy efficiency for iterative algorithms. In this case, it is not necessarily important for the solution to be extremely fast or to work well for different problems. However, there are times when different requirements need to be fulfilled. In these cases, there are a lot of parameters that can influence the performance of these tasks on the multitude of frameworks available [11]. This devolves the problem to one of determining which framework will most optimally solve some big data problem for a given set of input parameters.

In this paper, we propose a delegation system that automatically determines the best big data processing framework suited for a given big data problem based off of various input parameters. This delegation system takes a set of frameworks (e. g., Hadoop MapReduce, Hive, Spark, etc.) and a set of input parameters (problem size, cluster distribution, problem type, etc.) and calculates which framework will provide the optimal execution. All of this is done with minimal overhead and maximum accuracy using multiple linear regressions.

The proposed delegation system is useful for determining which framework is most suited for a given task and is a unique attempt at combining all of the various frameworks available through a single, unified interface. To validate our system, we focused on two big data processing frameworks, namely, Hadoop MapReduce and AJIRA [9]. Our experimental results show that the delegation system successfully determines which big data framework best suits a given task based off of problem and size parameters. An evaluation of the results also revealed that the overhead time was minimal when compared to the execution time of the actual code. These results offer an empirical demonstration of the usefulness of this system in determining the optimal framework for a given big data problem.

This paper is organized as follows. In Section 2, we give a background of work related to this field of study. Section 3 describes the reasoning behind the approach to this solution and section 4 expands upon that by providing implementation.
specific information. Section 5 details the results of our experiments and provides an in-depth analysis of those results. In Section 6, we discuss future work that can be done to improve upon the current interface. In Section 7, we provide concluding remarks.

II. RELATED WORK

Currently, there is a lot of research material related to improving Hadoop and creating various frameworks that improve upon Hadoop. Work has been done attempting to improve task scheduling in Hadoop and MapReduce. Althebyan et al. did work on improving MapReduce task scheduling by implementing a multi-threaded approach [4]. This improved energy efficiency and decreased wait times for programs across the board. Ren et al. improved task scheduling by prioritizing smaller tasks over larger tasks [6]. Data obtained from the Taobao cluster¹ suggested that the majority of tasks run were small tasks, and that improving the task scheduler to take advantage of this information would improve time to completion for most tasks.

In addition to improving the scheduling for Hadoop and MapReduce, substantial work has also been done on improving scalability. Appuswamy et al. did work on improving the performance of small applications by running MapReduce programs on more powerful localized machines [5]. Results showed that it was faster to scale-up for smaller tasks because it reduced the overhead incurred by Hadoop for distributing the data. Wang et al. performed tests on integrating Hadoop for HPC systems [8]. Modern HPC systems have high speed interconnects and lots of memory, so adjustments were made to Hadoop to gain increased performance on these systems.

There has also been effort put into improving network communication overhead costs through the use of new emerging routing technologies. Rose et al. created Pythia, a predictable software-defined networking model that dynamically calculates the optimal networking path of MapReduce programs based on network loads, bandwidth available and task locations on a cluster [13].

Work is also being done on improving big data frameworks and creating new frameworks to take advantage of extra computer components. Shinnar et al. performed tests on adding memory usage functionalities to MapReduce [7]. Results show that, for memory intensive problems, their model has significant performance boosts than standard MapReduce. Lee et al. created another SQL to MapReduce translator that improved upon previous works by automatically detecting intra-query correlations and spawning a minimal number of jobs to execute these tasks [14]. Results showed that Ysmart outperformed Hive and Pig by more than four times for query execution [14].

The above research shows that lots of work has been put into improving some facets of the Hadoop MapReduce framework. However, the area of creating a framework delegation tool seems to be unexplored, making this proposed solution unique.

III. DESIGN APPROACH

Work has been done on creating different tools with which big data can be processed. Some tools work best on specific problems while other tools have a broader purpose. However, no single tool is strictly better than others in all areas [11]. While it may currently be unfeasible to create the best big data analytics tool for all problems, we argue that it is possible to create an interface that chooses the best tool for a given job based on various conditions related to performance (architecture, cluster layout, node count, problem size, problem type, etc.). We propose a single interface to test this idea. For this problem in particular, we combined Hadoop’s MapReduce and AJIRA using an interface that computes the optimal program that best suits the task at hand. In the following sections, we give more specific details on the reasoning behind why these frameworks were chosen.

A. Hadoop

Hadoop is an open-source software for reliable, scalable, distributed computing [2]. Hadoop MapReduce is a big data framework that provides several unique features that explain its current prominence. These features are scalability, fault tolerance and flexibility [3,11]. It achieves all of these features through the implementation of a distributed file system (DFS). This feature adds overhead in terms of file management, but works best when used in parallel with lots of clustered machines on large-scale problems. However, it is precisely because of this initial overhead that Hadoop MapReduce does not optimally solve smaller problems.

B. AJIRA

AJIRA is a middleware designed for generic data processing developed by researchers at Vrije University, Amsterdam [9]. AJIRA was developed to improve upon Hadoop MapReduce by offering a more expressive toolset and by adding improved scalability through the use of memory accessibility. By focusing on these features, AJIRA’s developers were able to create a framework that was faster than Hadoop for problems of smaller sizes on ad-hoc clusters [9]. One of the few downsides to the current implementation is that it does not interface with DFSs making it ill-suited for solving larger problems efficiently [9]. This improved performance for smaller problem sizes contrasts with Hadoop MapReduce’s excellent ability at processing very large volumes of data. Our motivation in this work is to develop an interface, called delegation interface, that selects the software framework that yields the best performance for a given big data problem.

C. Combining Hadoop and AJIRA

The delegation interface determines at run time which code to execute (MapReduce or AJIRA) based on data obtained from previous execution patterns. Figure 1 provides a graphical view of the delegation process. Initially, the parameters for the current tasks are calculated. These parameters are user-defined and can be anything from program size to energy efficiency. There will be separate data locations for both Hadoop and AJIRA, each of which contains data on the problem type, execution size and other various parameters

¹ This is a 2,000-node Hadoop cluster at Taobao, which is the biggest online e-commerce enterprise in Asia, ranked 14th in the world as reported by Alexa [12].
(size, energy-used, node configuration, etc.). The interface will read information from these locations and create a linear regression formula representing each input script. The current task’s parameters will then be inserted as variables to the derived equations and an optimal framework will be calculated. Once the code is executed on the optimal framework, information regarding the current run will be inserted into the file for that respective framework. This data will then be used in the next iteration for the calculation of the regression formula. Using this approach, it is feasible to determine which framework is most optimal for a given problem with a certain set of defined parameters. Data storage can be any kind of persistent storage solution. In-memory storage solutions would provide the best performance possible, but it is also possible to use disk-based solutions and obtain fast calculation speed. This is due to the relatively small amount of data required for regression calculations.

IV. IMPLEMENTATION

The code that implements the design presented in the previous section was programmed in Java. This decision was made because of the advanced object oriented structure Java contains and because both Hadoop MapReduce and AJIRA were written in Java. The code takes four arguments: a name, an input file, a Hadoop script and an AJIRA script. The name is taken and used to create a storage location from which output files related to similar problems will be placed. Once these storage locations are created, the program naively runs the Hadoop script first to get some sample data. Information regarding the execution time of this problem on Hadoop is recorded and stored. Upon the next execution of this interface, the AJIRA script is naively run for the same reasons as before. This process is repeated k times, where k is the number of independent variables that are being recorded. This is all to get the k data instances required to create a linear regression that will be used to estimate future executions. Once the program has been run 2k times (k times for each problem), linear regression information is calculated and used to determine which framework the input problem should be executed on. Information is then gathered from that execution and added to the storage location. This process is repeated for subsequent executions of the same problem and restarted for execution of new problems. This algorithm runs in time O(k*t), where k is the number of data points available and t is the number of independent variables used in calculating the linear regression formula.

Currently, there is only one issue with the implemented code and that is due to the difference in storage requirements for the frameworks. Hadoop runs only on the DFS and AJIRA does not. Extra overhead time will be added to the Hadoop data if only script execution time is taken into account. Because of this, time calculations have to be done in script. This still leaves some overhead differences between dealing with a DFS and a local file system. The only way to truly put both frameworks on equal terms is to add DFS functionality to AJIRA. However, even with this minor difference, results showed that the combined interface improved over both frameworks in their less optimal areas. The code for our delegation system can be downloaded on github at the following address: [10].

Fig. 1. Delegation Interface Diagram
V. TESTING AND ANALYSIS

This section describes the test environment as well as the specific inputs and outputs obtained from each test. The final subsection provides a detailed analysis of the test results, which show support for the use of a delegation framework.

A. Testing Environment

All tests were run in a virtual box machine with 1 GB of RAM, dynamically sized disk space with a maximum of 100 GB, 4 Intel i7 cores and a 64-bit Ubuntu. We used Hadoop 2.5.2 and the latest version of AJIRA (downloaded from github as mentioned in [9]). Intermediate data was stored in the temporary directory of the virtual machine disk space.

B. Test Data

The test data used was a set of complete Wikipedia edit history: an 18 GB compressed, 3 TB uncompressed data set. This data was stored in the home directory of the Ubuntu machine disk space. This data set was chosen because of the size and variety of the data available. Using the 7zip software, it was possible to obtain samples of various sizes ranging from one megabyte to tens of gigabytes. Each of these data samples was used as a base standard for the AJIRA and Hadoop samples as well as testing files for the interface. The program that was run on this data is a word count program. Word count is used widely in the big data field as an introductory program to grasp how to program in various environments. It also serves as a simple test of performance for various frameworks.

C. Test Methodology

The goal of our delegation system is to automatically determine the most optimal framework to run for a given problem and a given set of problem parameters. Since Hadoop MapReduce and AJIRA excel at processing large and small data respectively, problem size was chosen as one of the parameters.

First, a baseline is calculated for both AJIRA and MapReduce executions of different problem sizes. Then, the delegation interface is run to determine which framework to select, how long it takes to execute on the chosen framework, and the extra time the interface adds to the total execution time. The interface uses the regression information and calculates the most optimal framework to execute while also bearing a minimal overhead. Input testing for the interface starts blank with no information known about the execution patterns. Information will be obtained during execution of standard jobs, which simulates a regular cluster job environment. Tests are performed so that small data tasks are executed before large data tasks. This is done to keep the testing environment consistent.

D. Test Results and Analysis

Figure 2 shows the results of the baseline tests. The x-axis represents a log scale of data size in megabytes and the y-axis is a log scale of the time taken for the given script to execute. AJIRA has a smaller linear regression intercept and a larger slope than the Hadoop regression. This confirms what was stated earlier, that Hadoop would perform better for larger problems and AJIRA would perform better for smaller problems. We can see from the figure that Hadoop starts to outperform AJIRA at around 100 megabytes.

Figure 3 shows the results of executing the interface program with the two AJIRA and Hadoop scripts as input. The axes are the same as in Figure 2. A red dot means that Hadoop was chosen to execute for the data size and a blue dot means AJIRA was chosen. As can be seen, the interface did correctly choose the most optimal solution for the larger test cases and came fairly close to choosing optimally for the smaller test cases. The reason the interface did not choose the correct framework for the smaller cases is due to the lack of information at the beginning of the execution process. The way the program is implemented, Hadoop is executed naively as the first program, then AJIRA as the second for whichever tasks are input. This is because nothing is known about the execution patterns at these initial stages of regression formula calculation. However, once more information is obtained by
running through a few jobs, the regression formulas converge and can be used to accurately calculate which framework is optimal. It should be noted that execution order is important. If the programs were executed from largest to smallest, then the incorrect choices would have been for larger data sizes. However, the formulas should still converge at a similar rate, becoming more accurate as data is collected. The data obtained in Figure 3 shows that the interface does determine an optimal framework when compared with execution times in Figure 2.

Figure 4 shows the overhead time taken by executing the interface as opposed to just directly running the scripts. The x-axis remains the same as before, but now the y-axis is in milliseconds. For tests of these sizes, interface overhead was at most 140 milliseconds, roughly 1% of the total execution time for the smallest problems and a paltry 0.01% of the execution time for the largest problems. It also shows that, aside from the initial scripts where linear regression was not performed due to incomplete data, the overhead time is roughly linear.

For testing purposes, local disk was used to store intermediate data. If a more efficient data storage solution, such as an in-memory database or SSDs, was used instead, the overhead would only decrease further.

VI. FUTURE WORK

There are many ways the current delegation interface can be expanded upon and improved. The current interface does not contain support for different intermediate data storage solutions. A feature can be added that allows the user to determine which storage should be utilized, whether that is an in-memory database or some other type of storage. This specific implementation only accounts for AJIRA and Hadoop scripts. Adding functionality to incorporate different frameworks is a top priority, as well as implementing a modular interface that allows for the easy addition of new frameworks. Currently, only size and program type are taken into account. A modular interface can be created that supports the inclusion of different input parameters, such as cluster configuration, energy efficiency and node count. This way new parameters can be easily added to existing data sets without requiring a complete overhaul of the system. Having multiple scripts representing the same code is a tedious and time wasting chore. It should be possible to create an interpreter that can take code from one language and map it to all others. This way, only one code would need to be written for a single framework. This would improve portability, but also increase the amount of overhead this program requires. More sophisticated regression analysis techniques could also be applied. Currently, redundant data is not accounted for (e.g., if two input programs have the same size, only the first is taken). Adding a method to take redundant data and combine it with other data could improve accuracy of regression calculations. Work could also be put into testing a problem extrapolation mechanism that determines similarities between tasks. This way, when a new problem is introduced, the interface could start with an initial guess estimate of which framework would be best for the given task as opposed to blindly picking frameworks until enough data is gathered.

VII. CONCLUSION

The proliferation of the Internet of things increases not only the amount of data created, but also the diversity of that data. Many different tools exist that excel in analyzing specific types of big data, but no tool exists that optimally analyzes all kinds of big data. In this paper, we presented a delegation system that aims at quickly choosing a tool given a set of selection parameters so that data can be analyzed as quickly as possible.
The proposed delegation system uses multiple linear regressions to determine which framework is best suited for a given task. The delegation tool reads in variables from a storage location and then analyzes that data to create a regression formula. Once the system has calculated which framework is most optimal, it executes the code on that framework and obtains information. The interface then stores this information for later use in the calculation of the regression formula.

Our evaluation shows that the system’s regression functions converge fairly quickly and that it can pick the optimal interface while maintaining a low overhead. This makes the delegation system an ideal tool for choosing which framework to utilize and optimize over a specific set of parameters. The proposed delegation system could represent an ideal unifying technology for all big data processing frameworks.

VIII. REFERENCES

[10] https://github.com/NickDavisIsCool/OMNIRA, code used in the development of this research project