MANTECH Publications
Recent Trends in Computer Science and Software Technology
Volume 3, Issue 1, January-April 2018

Analytics Process of Big Data Using Map Reduce

Rahul H Sathawane[1], Santosh Kumar Sahu[2]
Assistant Professor[1, 2]
Department of IT[1], Department of MCA[2]
Rajiv Gandhi College of Engineering and Research, Nagpur[1],
Shri Ramdeobaba College of Engineering and Management, Nagpur[2]
***Corresponding Authors' email id**: rahul.sathawane129@gmail.com[1]*

Abstract

Big data analytics is the process of examining large and varied data sets -- i.e., big data -- to uncover hidden patterns, unknown correlations, market trends; customer preferences and other useful information that can help organizations make more-informed business decisions. Big data is big news and so too is analytics on big data. Technologies for analyzing big data are evolving rapidly and there is significant interest in new analytic approaches such as Hadoop, MapReduce and Hive, and MapReduce extensions to existing relational DBMSs. This paper examines the benefits of big data and the role of the data scientist in deploying big data solutions. It discusses the benefits of Hadoop, MapReduce, Hive, and relational DBMSs that have added MapReduce capabilities. It presents a set of scenarios outlining how Hadoop and relational DBMS technology can coexist to provide the benefits of big data and big data analytics to the business and to data scientists. As an example of the use of MapReduce in a relational DBMS, the paper also reviews the implementation of MapReduce in the Aster Database

Keywords: *Data analytics, Data scientist, Big data, Map reduce, Hive*

I. INTRODUCTION

Big data analytics is the process of examining large data sets containing a variety of data types – i.e., big data -- to uncover hidden patterns, unknown correlations, market trends, customer

preferences and other useful business information. The analytical findings can lead to more effective marketing, new revenue opportunities, better customer service, improved operational efficiency, competitive advantages over rival organizations and other business benefits. The primary goal of big data analytics is to help companies make more informed business decisions by enabling data scientists, predictive modelers and other analytics professionals to analyze large volumes of transaction data, as well as other forms of data that may be untapped by conventional business intelligence(BI) programs.

The topic of big data is receiving significant press coverage and gathering increasing interest from both business users and IT staff. The result is that big data is becoming an overhyped and overused marketing buzzword. It remains, however, a valuable term because from an analytics perspective it still represents analytic workloads and data management solutions that could not previously be supported because of cost considerations and/or technology limitations.

These solutions can bring significant benefits to the business by enabling smarter and faster decision making, and allowing organizations to achieve faster time to value from their investments in analytical processing technology and products.

Smarter decision making comes from the ability to analyze new sources of data to enhance existing analytics and predictive models created from structured data in operational systems and data warehouses. There is strong emphasis in big data products, for example, on the analysis of multi-structured1 data from sensors, system and web logs, social computing web sites, text documents, and so forth. Hitherto, these types of data have been difficult to process using traditional analytical processing technologies.

Faster decisions are enabled because big data solutions support the rapid analysis of high volumes of detailed data. The analysis of large data stores has been difficult to date because it is often impossible to implement in a timely or cost-effective manner.

Faster time to value is possible because organizations can now process and analyze data that is outside of the enterprise data warehouse. It is often not practical or cost effective, for example, to integrate large volumes of machine-generated data from

sensors and system and web logs into the enterprise data warehouse for analysis. In many cases, however, analyzing machine-generated data helps identify smaller subsets of high-value information that can be integrated into the enterprise data warehouse.

To realize the full value of big data, organizations need to build an investigative computing platform2 that enables business users such as data scientists to ingest, structure and analyze big data to extract useful business information that is not easily discoverable in its raw native form.

2. THE ROLE OF DATA SCIENTIST

A role or job frequently associated with big data is that of the data scientist. Daniel Tunkelang, Principal Data Scientist at LinkedIn, states, "Data scientists turn big data into big value, delivering products that delight users, and insight that informs business decisions."[3] Data scientists use a combination of their business and technical skills to investigate big data looking for ways to improve current business analytics and predictive analytical models, and also for possible new business opportunities.

One of the biggest differences between a data scientist and a business intelligence (BI) user – such as a business analyst – is that a data scientist investigates and looks for new possibilities, while a BI user analyzes existing business situations and operations.

Data scientists require a wide range of skills:

- Business domain expertise and strong analytical skills
- Creativity and good communications
- Knowledgeable in statistics, machine learning and data visualizatio
- Able to develop data analysis solutions using modeling/analysis methods and languages such as MapReduce, R, SAS, etc.
- Adept at data engineering, including discovering and mashing/blending large amounts of data

People with this wide range of skills are rare, and this explains why data scientists

are in short supply. In most organizations, rather than looking for individuals with all of these capabilities, it will be necessary instead to build a team of people that collectively has these skills.

Data scientists use an investigative computing platform (see Figure 2.1) to bring unmodeled data, e.g., multi-structured data, into an investigative data store for experimentation. This data store is separated, but connected to, the traditional data warehousing environment. In some cases, existing modelled data from the data warehouse may also be added to the data store.

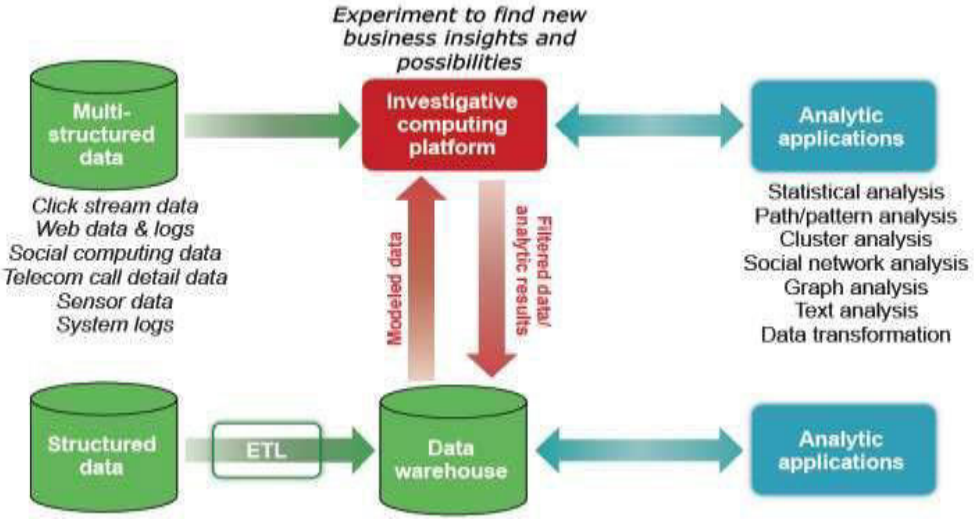

Figure 2.1 Investigative computing

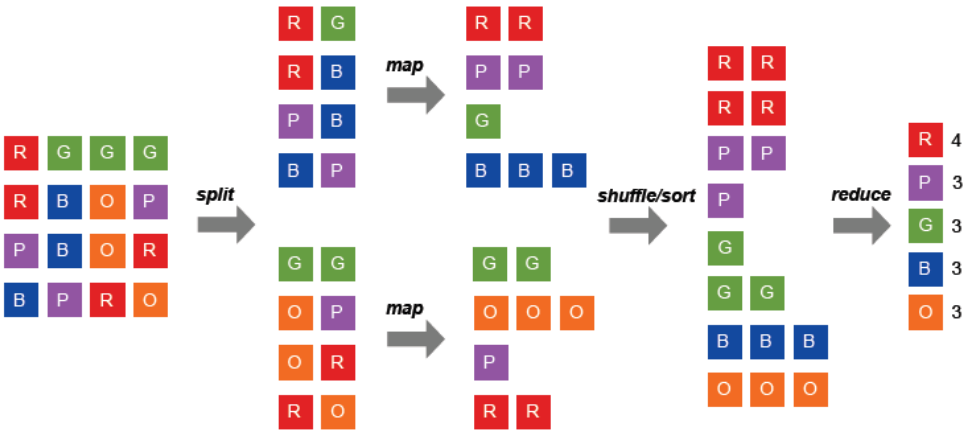

Figure 3.1 Colored Square Counter

3. WHAT IS MAPREDUCE?

MapReduce is a technique popularized by Google that distributes the processing of very large multi-structured data files across a large cluster of machines. High performance is achieved by breaking the processing into small units of work that can be run in parallel across the hundreds, potentially thousands, of nodes in the cluster.

To quote the seminal paper on MapReduce:

"MapReduce is a programming model and an associated implementation for processing and generating large data sets. [12]Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system." The key point to note from this quote is that MapReduce is a programming model, not a programming language, i.e., it is designed to be used by programmers, rather than business users. The easiest way to describe how MapReduce works is through the use of an example – see the Colored Square Counter in Figure 3.1.

The MapReduce system first reads the input file and splits it into multiple pieces. In this example, there are two splits, but in a real life scenario, the number of splits would typically be much higher. These splits are then processed by multiple map programs running in parallel on the nodes of the cluster. The role of each map program in this case is to group the data in a split by color. The MapReduce system then takes the output from each map program and merges (shuffle/sort) the results for input to the reduce program, which calculates the sum of the number of squares of each color. In this example, only one copy of the reduce program is used, but there may be more in practice. To optimize performance, programmers can provide their own shuffle/sort program and can also deploy a combiner that combines local map output files to reduce the number of output files that have to be remotely accessed across the cluster by the shuffle/sort step.

4. WHY USE MAPREDUCE?

MapReduce aids organizations in processing and analyzing large volumes of multistructured data. Application examples include indexing and search, graph analysis, text analysis, machine learning, data transformation, and so forth. These types of applications are often difficult to

implement using the standard SQL employed by relational DBMSs.

The procedural nature of MapReduce makes it easily understood by skilled programmers. It also has the advantage that developers do not have to be concerned with implementing parallel computing – this is handled transparently by the system. Although MapReduce is designed for programmers, non-programmers can exploit the value of prebuilt MapReduce applications and function libraries.

Both commercial and open source MapReduce libraries are available that provide a wide range of analytic capabilities. Apache Mahout, for example, is an open source machine-learning library of "algorithms for clustering, classification and batch-based collaborative filtering" that are implemented using MapReduce.

4.1 The How of Mapreduce: Application Development

MapReduce programs are usually written in Java, but they can also be coded in languages such as C++, Perl, Python, Ruby, R, etc. These programs may process data stored in different file and database systems. At Google, for example, MapReduce was implemented on top of the Google File System (GFS). One of the main deployment platforms for MapReduce is the open source Hadoop distributed computing framework provided by Apache Software Foundation. Hadoop supports MapReduce processing on several file systems, including the Hadoop Distributed File System (HDFS), which was motivated by GFS. Hadoop also provides Hive and Pig, which are high-level languages that generate MapReduce programs. Several vendors offer open source and commercially supported Hadoop distributions; examples include Cloudera, DataStax, Hortonworks (a spinoff from Yahoo) and MapR. Many of these vendors have added their own extensions and

Modifications to the Hadoop open source platform

Another direction of vendors is to support MapReduce processing in relational DBMSs. These are implemented as in-database analytic functions that can be used in SQL statements. These functions are run inside the database system, which enables them to benefit from the parallel processing capabilities of the DBMS. Supported in the Teradata Aster MapReduce Platform, the Aster Database provides a number of Built-in MapReduce functions for use with SQL. It also

includes an interactive development environment, Aster Developer Express, for programmers to create their own MapReduce functions.

5. USING HIVE FOR MAPREDUCE DEVELOPMENT

Hive is one of several high-level languages for developing MapReduce applications. Two others are Pig and JAQL.7 Pig was developed by Yahoo and is now a component of the Apache Hadoop project. It includes Pig Latin, which is a high-level data flow and manipulation language for building MapReduce applications. It has both procedural (like a programming language) and declarative (as in SQL) statements and is somewhat similar to (but less powerful) than early fourth generation languages such as Focus, Nomad and SAS.[3] JAQL is a query language designed for Javascript Object Notation (JSON), a data format that has become popular because of its simplicity. JAQL was developed by IBM for its Big Insights product. Although IBM has placed JAQL in the open source community, it has gained little traction outside of the IBM environment.

This paper focuses on Hive because it has gained the most acceptance in the industry and also because its SQL-like syntax makes it easy to use by non-programmers who are comfortable using SQL. Many of the conclusions in this paper about Hive, however, are equally applicable to Pig, JAQL and other high-level languages used for deploying MapReduce solutions.

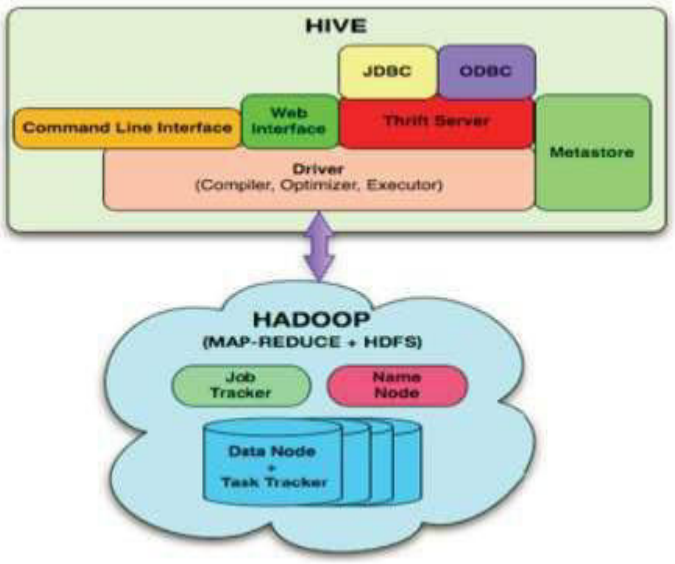

Figure 3.1: Hive components

The main components of Hive are illustrated in Figure 3. HiveQL statements can be entered using a command line or Web interface, or may be embedded in applications that use ODBC and JDBC interfaces to the Hive system. The Hive Driver system converts the query statements into a series of MapReduce jobs.

Data files in Hive are seen in the form of tables (and views) with columns to represent fields and rows to represent records. Tables can be vertically partitioned based on one or more table columns. The data for each partition is stored in a separate HDFS file. Data is not validated against the partition definition during the loading of data into the HDFS file. Partitions can be further split into buckets by hashing the data values of one or more partition columns. Buckets are stored in separate HDFS files and are used for sampling and for building a Hive index on an HDFS file. Hive tables do not support the concepts of primary or foreign keys or constraints of any type. All definitions are maintained in the Hive metastore, which is a relational database such as MySQL.

Data types supported by Hive include primitive types such as integers, floating point numbers, strings and boolean. A timestamp data type is provided in Hive 1.8.1. Hive also supports complex types such as arrays (indexed lists), maps (key value pairs), structs (structures) and user-defined types. External HDFS files can be defined to Hive as external tables. Hive also allows access to data stored in other file and database systems such as HBase. [11]Access to these data stores is enabled via storage handlers that present the data to Hive as non-native tables. The HiveQL support (and restrictions) for these non-native tables is broadly the same as that for native tables. HiveQL supports a subset of the SQL SELECT statement operators and syntax, including:

- Join (equality, outer, and left semi-joins are supported)
- Union
- Subqueries (supported in the FROM clause only)
- Relational, arithmetic, logical and complex type operators
- Arithmetic, string, date, XPath, and user-defined functions including aggregate and table functions (UDFs, UDAFs, UDTFs)

The Hive MAP and REDUCE operators can be used to embed custom MapReduce scripts in HiveQL queries. An INSERT

statement is provided, but it can only be used to load or replace a complete table or table partition. The equivalents of the SQL UPDATE and DELETE statements are not supported. When comparing Hadoop Hive to a relational DBMS employing SQL, two areas have to be considered: query language syntax and query performance. Query language syntax is a moving target in both Hadoop Hive and relational DBMS products. Although Hive provides a useful subset (and superset) of SQL functionality, it is highly probable that existing SQL-based user applications and vendor software would need to be modified and simplified to work with Hive.

Perhaps the most important comparison between Hadoop Hive and the relational DBMS environment concerns performance. Such comparisons should consider traditional short and/or ad-hoc SQL-like queries running on Hive versus a relational DBMS, and also MapReduce performance on Hive compared with using SQL MapReduce relational DBMS functions for querying and analyzing large volumes of multi-structured data. Knowledge of the way Hive and relational DBMSs process queries is useful when discussing the performance of the two approaches.

Hive provides an SQL wrapper on top of Hadoop HDFS (and other storage systems).It has no control or knowledge of the placement or location of data in HDFS files. The Hive optimizer uses rules to convert HiveQL queries into a series of MapReduce jobs for execution on a Hadoop cluster. Hints in HiveQL queries can aid the optimization process, for example, to improve the performance of join processing.

Hive-generated MapReduce jobs sequentially scan the HDFS files used to store the data associated with a Hive table. The Hive optimizer is partition and bucket aware, and so table partitions and buckets can be used to reduce the amount of data scanned by a query. [5]Hive supports compact indexes (in 0.7.0) and bitmapped indexes (in 0.8.0), which aid lookup and range queries, and also enable certain queries to be satisfied by index-only access. Since Hive has no knowledge of the actual physical location of the data in an HDFS file, the Hive indexes contain data rather than pointers to data. A table index will need to be rebuilt if the data in a table partition is refreshed. Hive does, however, support partitioned indexes. Hive indexes aid the performance of traditional SQL-like queries, rather than MapReduce

queries, which by their nature involve sequential processing.

The primary use case for Hadoop Hive is the same as that for Hadoop MapReduce, which is the sequential processing of very large multi-structured data files such as Web logs. It is not well suited to ad- hoc queries where the user expects fast response times. The positioning of Hive is aptly described on the Apache Hive Wiki. "Hive is not designed for OLTP workloads and does not offer real-time queries or row-level updates. It is best used for batch jobs over large sets of append-only data (like Web logs). What Hive values most is scalability (scale out with more machines added dynamically to the Hadoop cluster), extensibility (with MapReduce framework and UDF/UDAF/UDTF), fault-tolerance, and loose-coupling with its input formats."

The main benefit of Hive is that it dramatically improves the simplicity and speed of MapReduce development. The Hive optimizer also makes it easier to process interrelated files as compared with hand-coding MapReduce procedural logic to do this. The Hive optimizer, however, is still immature and is not fully insulated from the underlying file system, which means that for more complex queries, the Hive user is still frequently required to aid the optimizer through hints and certain HiveQL language constructions.

There are many other tools for improving the usability of Hadoop, e.g., Informatica HParser for data transformation, and Karmasphere Studio, Pentaho Business Analytics and Revolution RevoConnectR for analytical processing. Most of these tools are front ends to Hadoop MapReduce and HDFS, and so many of the considerations discussed above for Hive apply equally to these products.

5. RELATIONAL DBMS MAPREDUCE DEVELOPMENT

Before discussing how relational DBMSs support MapReduce, it is useful to first compare how Hadoop Hive and a relational DBMS process queries. This process is illustrated in Figure 5.

The left side of the diagram shows the operation of Hadoop Hive outlined in the previous section. The Hive rules-driven optimizer processes HiveQL statements and generates one or more MapReduce jobs to carry out the required query operations.

These jobs are passed to the Hadoop system, where they are scheduled and managed by Hadoop's job and task tracker

capabilities. The MapReduce jobs read and process HDFS data and pass the results back to the Hive application. With the SQL application shown on the right side of the diagram, the relational mapping layer provides an interface between the relational logical view of data and the data management software used to create and manipulate physical data in the storage subsystem.[7]

In general, the logical and physical views of data in a relational system are completely isolated from each other in order to provide physical data independence. As discussed earlier, the isolation in Hadoop Hive is not as complete, which reduces data independence, but may give the developer more control for certain complex queries.

The data independence of a relational DBMS has the advantage that vendors can extend or add runtime and data storage engines without affecting existing applications. [10]The open source MySQL relational DBMS, for example, offers several data storage engines. Commercial vendors have also extended and added new data storage engines to support XML data, online analytical processing (OLAP), and so forth. The Teradata Aster Database includes a runtime engine for MapReduce processing, as well as a data store that enables both row and columnar data storage.

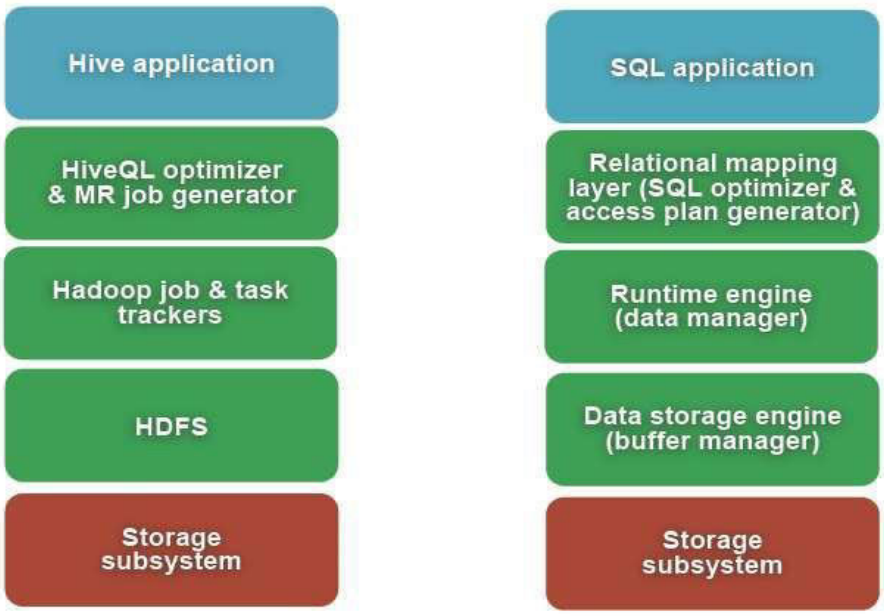

Figure 5.1 Hadoop Hive and relational DBMS query processing

CONCLUSIONS

Big data and associated technologies can bring significant benefits to the business, but as the use of these technologies grows, it will become difficult for organizations to tightly control all of the many forms of data used for analysis and investigation. It will certainly become impossible for organizations to centralize all of the data in a single enterprise data warehouse (EDW) and it will become necessary to distinguish between data that can, and that cannot, be consolidated into the EDW. For data that remains outside of the EDW environment (for performance or cost reasons, for example), BI developers and data scientists should evaluate the best approach to use for filtering and/or analyzing the data. Options are to use a relational DBMS optimized for analytic processing such as the Aster Database, a non-relational system such as Hadoop with Hive, or to analyze in-motion data as it flows through enterprise systems using data streaming technology.

For EDW data it will be necessary to determine the data that needs to be extracted into an underlying data mart or data cube for analysis and reporting by BI users, or into an investigative data sandbox for investigation by data scientists. This latter data sandbox will likely also contain multi-structured data, possibly extracted from a Hadoop system. Given these many different approaches and components, organizations must extend their existing EDW infrastructure to support them. This new infrastructure can be thought of as an extended data warehouse. Such an infrastructure is essential if organizations are to reap the full benefits of big data, and enable data scientists to investigate ways of improving the business and also identify new business opportunities.

REFERENCES

I. A. Gates and S. Babu. Profiling, Cost-based Optimization of MapReduce Programs.PVLDB,4(11):1111–1122, 2012.

II. D. Abadi et al. Column-Oriented Database Systems. PVDLB2(2):1664–1665, 2009.

III. F. N. Afrati and J. D. Ullman. Optimizing Joins in a Map-Reduce Environment. In EDBT, pages 99–110, 2010.

IV. S. Babu. Towards automatic optimization of MapReduce programs. In SOCC , pages 137–142, 2010.

V. S. Blanas et al. A Comparison of Join Algorithms for Log Processing in MapReduce. In SIGMOD , pages 975–986, 2010.

VI. J. Dean and S. Ghemawat. MapReduce: A Flexible Data Processing Tool. CACM , 53(1):72–77, 2010.

VII. J. Dittrich, J.-A. Quiane-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). PVLDB , 3(1):519–529, 2010.

VIII. J. Dittrich, J.-A. Quian e-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad. Only Aggressive Elephants are Fast Elephants. PVLDB , 5, 2012.

IX. A. Floratou et al. Column-Oriented Storage Techniques for MapReduce. PVLDB , 4(7):419–429, 2011.

X. A. Gates et al. Building a HighLevel Dataflow System on Top of MapReduce: The Pig Experience. PVLDB, 2(2):1414–1425, 2009.

XI. S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In SOSP, pages 29–43, 2003.

XII. H. Herodotou and S. Babu. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. PVLDB,4(11):1111–1122, 2011.

XIII. M. Isard et al. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In EuroSys, pages 59–72, 2007