IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

Handling Cassandra Datasets with Hadoop-Streaming

Akash Suryawanshi[1], Pratik Patil[2]

Computer Engineering B.V.D.U.C.O.E., Pune, India[1]

Information Technology B.V.D.U.C.O.E., Pune, India[2]

Abstract: The dynamic change in the idea of both logical and mechanical datasets has been the main impetus behind the advancement and research interests in the NoSQL display. Inexactly organized information represents a test to conventional information store frameworks, and when working with the NoSQL demonstrates, these frameworks are regularly viewed as illogical and exorbitant. As the amount and nature of unstructured information develops, so does the interest for a preparing pipeline that is able to do consistently joining the NoSQL stockpiling model and a "Major Data" handling stage for example, MapReduce. In spite of the fact that MapReduce is the worldview of decision for information serious processing, Java-based systems such as Hadoop expect clients to compose MapReduce code in Java while Hadoop Streaming module enables clients to characterize non Java executables as guide and lessen operations. Whenever stood up to with inheritance C/C++ applications and other non-Java executables, there emerges a further need to permit NoSQL information stores get to the highlights of Hadoop Streaming. We introduce approaches in comprehending the test of coordinating NoSQL information stores with MapReduce under non-Java application situations, alongside points of interest and drawbacks of each approach. We look at Hadoop Streaming nearby our own particular spilling system, MARISSA, to indicate execution ramifications of coupling NoSQL information stores like Cassandra with MapReduce structures that typically depend on document framework based information stores. Our trials additionally incorporate Hadoop-C*, which is where a Hadoop group is co-situated with a Cassandra group keeping in mind the end goal to process information.

Keywords: Cassandra, HADOOP.

I. INTRODUCTION

With the expanded measure of information gathering occurring because of web-based social networking communication, logical investigations, what's more, even web based business applications, the nature of information as we know it has been advancing. Because of this information age from a wide range of sources, "new age" information, presents challenges as it isn't all social and needs predefined structures. For instance, blog segments for business elements gather different contributions from clients about their items from Twitter, Facebook, and web-based social networking outlets. Be that as it may, the structure of this information contrasts incomprehensibly on the grounds that it is gathered from differed sources. A comparable marvel has emerged in the logical field, for example, at NERSC where information originating from a solitary investigation may include different sensors observing unique parts of a given test. In this condition, information significant to that investigation all in all will be created, yet it might be designed in various routes since it begins from diverse sources. While comparable difficulties existed before the appearance of the NoSQL show, prior methodologies included putting away in an unexpected way organized information in particular databases, and in this manner dissecting each dataset in segregation, possibly missing a "greater picture" or basic connection between datasets. Presently, NoSQL offers an answer for this issue of information seclusion by permitting datasets, having a similar setting yet not a similar structure or, on the other hand design, to be gathered together. This permits the information not just to be put away in similar tables however to therefore be broke down all in all. At the point when non-uniform information develops to extensive sizes be that as it may, a appropriated way to deal with investigate unstructured information needs to be considered. MapReduce has developed as the model of decision for preparing "Huge Data" issues. MapReduce structures, for example, Hadoop offer both stockpiling and preparing abilities for information in any shape, organized or not. Be that as it may, they don't straightforwardly offer help for questioning the information. Developing datasets not just should be questioned to empower genuine time data accumulation and sharing, yet additionally need to experience complex bunch information examination operations to remove the most ideal information. NoSQL information stores offer not just the capability of putting away huge, approximately organized information that can later be dissected and ned overall, however they likewise offer the capacity for inquiries to be connected on such information. This is particularly gainful when continuous answers are required on just cuts of the put away information. In spite of the nearness of this profitable cluster handling potential from NoSQL stores, there is a requirement for a product pipeline permitting "Huge Data" handling models, for example, MapReduce to tap NoSQL information stores as wellsprings of information. There is moreover a requirement for a product pipeline permitting

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

MapReduce heritage programs written in C, C++, and non-Java executables to utilize "Enormous Data" innovations.
In this paper, we present a processing pipeline allowing not only native Hadoop MapReduce programs written in Java to make use of the NoSQL storage systems, but also any nonJava executable used for data processing. Such a pipeline is useful in applications that require an offline batch processing platform. For example, Netflix is known to use data pipeline, which includes Apache Cassandra, for offline processing. We use Apache Cassandra in our analysis, a well-known NoSQL solution, and combine it with both Apache Hadoop and a MapReduce solution of our own MARISSA. We show on a case-by-case basis when it is beneficial to process the data directly from NoSQL stores and the performance impact of first downloading it to a MapReduce framework.

II. BACKGROUND

A. Cassandra is a non-relational and column-oriented, distributed database. It was originally developed by Facebook. It is now an open source Apache project. Cassandra is designed to store large datasets over a set of commodity machines by using a peer-to-peer cluster structure to promote horizontal scalability. In the column-oriented data model of Cassandra, a column is the smallest component of data. Columns associated with a certain key constitute a row. Each row can contain any number of columns. A collection of rows forms a column family, which is similar to tables in relational databases. Records in the column families are stored in sorted order by row keys, in separate files. The keyspace congregates one or more column families in the application, similar to a schema in a relational database. Interesting aspects of the Cassandra framework include independence from any additional file systems like HDFS, scalability, replication support for fault tolerance, balanced data partitioning, and MapReduce support with a Hadoop plug-in. In, we present a detailed discussion on the attributes of Cassandra that make it interesting for MapReduce processing.

B. MapReduce Taking inspiration from functional programming, MapReduce starts with the idea of splitting an input dataset over a set of commodity machines, called workers, and processes these data splits in parallel with user-defined map and reduce functions. MapReduce abstracts away from the application programmers the details of input distribution, parallelization, scheduling and fault tolerance.

1) Hadoop & Hadoop Streaming: Apache Hadoop is the leading open source MapReduce implementation. Hadoop relies on two fundamental components: the Hadoop Distributed File System (HDFS) and the Hadoop MapReduce Framework for data management and job execution respectively.
A Hadoop Job Tracker, running on the master node is responsible for resolving job details (i.e., number of mappers/reducers), monitoring the job progress and worker status. Once a dataset is put into the HDFS, it is split into data chunks and distributed throughout the cluster. Each worker hosting a data split runs a process called DataNode and a TaskTracker that is responsible for processing the data splits owned by the local DataNode. Hadoop is implemented in Java and requires the map and reduce operations to also be implemented in Java and use the Hadoop API. This creates a challenge for legacy applications where it may be impractical to rewrite the applications in Java or where the source code is no longer available. Hadoop Streaming is designed to address this need by allowing users to create MapReduce jobs where any executable (written in any language or script) can be specified to be the map or reduce operations. Although Hadoop Streaming has a restricted model , it is commonly used to run numerous scientific applications from various disciplines. It allows domain scientists to use legacy applications for complex scientific processing or use simple scripting languages that eliminate the sharp learning curve needed to write scalable MapReduce programs for Hadoop in Java. Protein sequence comparison, tropical storm detection, atmospheric river detection and numerical linear algebra are a few examples of domain scientists using Hadoop Streaming on NERSC systems.

2) MARISSA: In earlier work, we highlighted both the performance penalty and application challenges of Hadoop Streaming and introduced MARISSA to address these shortcomings. MARISSA leaves the input management to the underlying shared file system to solely focus on processing. In we explain the details of MARISSA and provide a comparison to Hadoop Streaming under various application requirements. Unlike Hadoop Streaming, MARISSA does not require processes like TaskTrackers and DataNodes for execution of MapReduce operations. Once the input data is split by the master node using the Splitter module and placed into the shared file system, each worker node has access to the input chunks awaiting execution.

III. MAPREDUCE STREAMING OVER CASSANDRA DATA

MapReduce Streaming Pipeline For Cassandra Datasets

we introduce a MapReduce pipeline that can be used by MapReduce frameworks like Hadoop Streaming and MARISSA that offer MapReduce capabilities with nonJava executables. This pipeline has three main stages: Data

Preparation, Data Transformation (MR1) and Data Processing (MR2). Each of these stages is explained in detail in the following subsections and performance implications are discussed in Section

1) Data Preparation: Data Preparation is the step where input datasets are downloaded from Cassandra servers to the corresponding file system – HDFS for Hadoop Streaming and the shared file system for MARISSA. In both of these frameworks, this step is initiated in parallel. Cassandra allows exporting the records of a target dataset in JSON formatted files and using this built-in feature each node downloads the data from the local Cassandra servers to the file system. In our experimental setup, each node that is running a Cassandra server is also a worker node for the MapReduce framework in use. Our experimental data has 3 columns. This choice aims to mimic storing Twitter user interaction logs in Cassandra. For 64 million records, we have 20GB data distributed through Cassandra servers with the replication factor set to 1. We implemented a set of tools to launch the process of exporting data from a Cassandra cluster. For each write request, Cassandra creates a commit log entry and writes mutated columns to an in-memory structure called Memtable. This inmemory structure is written into an immutable data file named SSTable at a certain size limit or predefined period of time. In our implementation, each worker connects in parallel to its local database server and flushes Memtables into SSTables After flushing data, workers begins the exporting operations. Every worker collects the exported records in unique files stored on the shared file system. In MARISSA, we were able to introduce these tools into the Splitter module. For Hadoop Streaming, however, we implemented additional tools to initiate the data preparation process in parallel on all worker nodes. Next, this data was placed into the HDFS using the put command from the Hadoop master node. In Hadoop's case, the put operation includes splitting the input into chunks and placing those chunks throughout the HDFS cluster. In MARISSA, however, the worker nodes flush the data to the shared file system and later these data files are split by the master one-by-one for each core. We compare the performance of the Data Preparation stage for Hadoop Streaming and MARISSA in Section IV-

2) Data Transformation (MR1): In the Data Preparation stage the input dataset from Cassandra servers is downloaded and placed into the shared file system or HDFS in JSON format. Moving the input dataset out of the database and into the file system also requires the exported data to be transformed into a format that can be processed by the target non-Java applications. Cassandra allows users to export its dataset as JSON formatted files. As our assumption is that the target executables are legacy applications which are either impossible or impractical to be modified, the input data needs to be converted into a format that is expected by these target applications. For this reason, our software pipeline includes a MapReduce stage, where JSON data can be transformed into other formats. This phase simply processes each input record and converts it to another format, writing the results into the intermediate output files. This step does not involve any data or processing dependencies between nodes and therefore is a great fit for the MapReduce model. In fact, we only initiate the map stage of MapReduce since no reduce operations are needed. If necessary for the conversion of JSON files to the proper format, a reduce step may be added conveniently. We implemented this stage in Python scripts that can be run using either MARISSA and Hadoop Streaming without any modifications. As this is the first of a series of iterative MapReduce operations whose output will be used as the input by the following MapReduce streaming operations, we simply call this stage MR1. Our system not only allows users to convert the dataset into the desired format but also makes it possible to specify the columns of interest. This is especially useful when only a vertical subset of the dataset is sufficient for the actual data processing. This stage helps to reduce data size, in turn affecting the performance of the next MapReduce stage in a positive manner. This performance gain is a result of fewer I/O and data parsing operations. In the following sections of this paper we will refer to this stage either as MR1 or as Data Transformation. Section IV- provides a comparison between the performance of Data Transformation using MARISSA and Hadoop Streaming.

3) Data Processing (MR2): This is the final step of the MapReduce Streaming pipeline shown in Figure 1. In Figure 1c we run the non-Java executables, which were the initial target applications, over the output data of MR1, as the data is now in a format that can be processed. We use MARISSA and Hadoop Streaming to run executables as map and reduce operations. Since this is the second MapReduce stage in our pipeline we name it MR2. Any MapReduce streaming job being run after MR1 is considered an MR2 step. In Section IV-C, we first compare the performance of MARISSA and Hadoop Streaming based only on this stage under various application scenarios. Later, in order to show the full operation span we include the time taken for Data Preparation and Data Transformation under each MapReduce framework and repeat our comparisons.

MapReduce Streaming Pipeline with MARISSA

As the Splitter module of MARISSA has been modified such that each worker connects to the local database server to take a snapshot of the input dataset in JSON format and place it into the shared file system. After the Data Preparation stage shown in Figure 1a the input is split and ready for Data Transformation. Figure 2a shows the architecture of MARISSA. It allows each non-Java executable to interact with the corresponding input splits directly without needing to mediate this interaction. In the stage of Data Transformation, each MARISSA mapper runs an executable to convert the JSON data files to the user-specified input format. These converted files are placed back into the shared file system.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

MARISSA runs the user given executables to create the next MapReduce stage, which we call Data Processing. This is accomplished using the previous output as the input. There is no re-distribution or re-creation of splits required since MARISSA is designed to allow iteration of MapReduce operations where the output of one operation is fed as input to the next.
MapReduce Streaming Pipeline with Hadoop Streaming In the Data Preparation stage, each Hadoop worker connects to the local Cassandra server and exports the input dataset in JSON formatted files. Next, these files are placed into the HDFS using the put command. This distributes the input files among the Data Nodes of the HDFS and later they are used as input for the Data Transformation stage. HDFS is a non-POSIX compliant file system that requires the use of HDFS API to interact with the files. the assumption is that these executables do not use this API and therefore do not have immediate access to the input splits. So, Hadoop Task Trackers read the input from HDFS and feed into the executables for processing and collect the results to write back to the HDFS. In the Data Transformation step shown in Figure 1b, Hadoop Streaming uses our input conversion code to transform the input to the desired format and later Data Processing is performed on the output of this stage. Note that at the Data Processing stage the input is already in HDFS as it is the output of the previous MapReduce job.

IV. PERFORMANCE RESULTS

The following experiments were performed on the Grid and Cloud Computing Research Lab Cluster at Binghamton University. • 8 Nodes in a cluster connected via Gigabit Ethernet, each of which has two 2.6Ghz Intel Xeon CPUs, 8 GB of RAM, 8 cores, Dual Local 73GB 15K RPM RAID Striped SAS Drive, and 64-bit version of Linux 2.6.15. • Single headless file server connected via 10GB Fiber Ethernet and Gigabit Ethernet, which has four 2.0Ghz Intel Xeon CPUs, 128 GB of RAM, 8 cores, 6.0 TB SAS RAID, provides NFSv4 and run a 64-bit version of Linux 2.6.15. • Apache Cassandra version 1.1.6 installed on each Hadoop slave nodes. • Hadoop version 1.2.1 installed on each node. In Table I and Table II, we present the important configuration parameters that are likely to affect the performance. In addition to these parameters, for both Cassandra and Hadoop, we used the default parameters shipped with specified distributions.

A. Data Preparation

the performance for taking a snapshot of the input dataset from Cassandra into the shared file system and HDFS for processing with MARISSA and Hadoop Streaming respectively. The cost of moving data from Cassandra servers expectedly increases with growing data sizes. Moving 256 million input records takes nearly 50 times more than moving 4 million. Figure 3 also shows the disparity for the cost of Data Preparation for Hadoop Streaming and MARISSA. At four million records Data Preparation for Hadoop Streaming is 1.1 times faster than MARISSA and it is over 1.2 times faster at 64 and 256 million records. This performance variation of Data Preparation for each system can be explained with the inefficiencies, laid out in Section III-A1, of the MARISSA Splitter module which is responsible for creating the data splits for individual cores of the worker nodes. We plan to address this inefficiency in MARISSA in future work.

B. Data Transformation (MR1)

The performance of Data Transformation which is the stage for converting the snapshot of the objective dataset to the required format using both MARISSA and Hadoop Streaming. This figure displays that MARISSA is almost eight times faster than Hadoop Streaming at four million input records but with growing data sizes this performance difference lessens. At 32 million records MARISSA performs 74 percent faster and with the further data sizes it keeps an advantage around 20 percent. We explain this change in performance gap between two frameworks in previous papers, that Hadoop start-up cost is more visible in small data sizes and the performance gets better for larger input datasets where this overhead is amortized for larger input sizes. On the other hand MARISSA framework is designed as a lightweight MapReduce platform where the start-up overhead in minimal.

C. Data Processing (MR2) In this section we run the target non-Java applications on the output of MR1. In the following tests we show the performance of running various applications scenarios in stage MR2 with MARISSA and Hadoop Streaming. In addition, we compare these two MapReduce streaming pipelines with Hadoop-C* where there is no Data Preparation and/or Data Transformation steps necessary and target application in MR2 can be run directly. We first show the performance of MR2 exclusively with each of the setups (MARISSA, Hadoop Streaming and Hadoop-C*) and later, in order to show the overall cost, with the times for Data Preparation, MR1 and MR2 combined..

1) Cluster Size Upgrade : Write Intensive Applications: Figure 9 compares the performance of running non-Java write intensive executables where the size of the output is roughly the same or larger than the input data. We ran experiments on 64 million input records with four different cluster sizes. In Figure 9a we show performance difference between jobs for Data Processing(MR2) stage for Hadoop Streaming. Since the Data Transformation(MR1) step provides great reduction in data size, running Data Processing stage on transformed data with Hadoop Streaming

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 11, November 2017

performs 35 times faster than Hadoop-C* and 16 times faster than Hadoop-C*-FS for cluster size of just two nodes. When we increase the cluster size, we observe that the performance disparity between Hadoop Streaming and Hadoop-C* setups decreases. In this case, the startup cost of each map task for Hadoop Streaming determines the Data Processing(MR2) time. Since the input data is distributed across the cluster, the number of data blocks increase. Accordingly, Hadoop Streaming schedules more map tasks. When the cluster is expanded from 6 to 8 nodes, over-reduced splits per mapper amortizes the benefit of increasing mapping capacity and causes Hadoop Streaming to keep similar execution time. Figure 9b includes overall execution time for Hadoop Streaming pipeline under various cluster sizes. For cluster size of 2, Hadoop-C*-FS performs 2.2 times faster than Hadoop-C* and 1.8 times faster than Hadoop Streaming. As we increase the number of nodes to 8, performance difference between Hadoop-C*-FS and other setups decreases by 10 percent for Hadoop-C* and by 15 percent for Hadoop Streaming. If we expand the cluster size, Hadoop Streaming benefits from the larger cluster in Data Preparation and Data Transformation(MR1) stages. As the number of nodes increases, amount of data exported from Cassandra servers gets smaller. Since the data size per node decreases, each worker node transforms exported data 3.2 times faster for 8 nodes compared to the 2 node setup. Increasing the cluster size expectedly reduces the execution time for fixed input records. Upon increasing the number of nodes from 2 to 4 HadoopC* and Hadoop-C*-FS runs 1.7 times faster and Hadoop Streaming runs 1.8 times faster. However, for Hadoop-C*-FS and Hadoop Streaming, doubling the cluster size from 4 to 8 does not provide the same speed up as when the cluster size was increased from 2 to 4. Speed up decreases by 15 percent Hadoop-C*-FS and 10 percent for Hadoop Streaming. On the other hand, Hadoop-C* shows the same speed up if we increase the number of nodes in the cluster. Memory allocated for in-memory writes for Cassandra cluster increases as we increase the number of nodes in the cluster. As Cassandra flushes Memtables into SSTables, at a certain threshold the number of flushes, which result in writes to the disk, performed at each node decreases.

2) Cluster Size Upgrade: Read Intensive Applications: Figure 10 displays performance of running read intensive nonJava executables on 64 million input records with different cluster sizes. Read intensive applications have a significantly larger input data size for MapReduce jobs compared to the output they generate. Therefore, since the output data is much smaller, Hadoop-C* and Hadoop-C-FS* show similar performance trends. So, we only consider performance results for Hadoop-C*. We focus on read performance with increasing number nodes regardless of writing performance to various targets.

V. PERFORMANCE SUMMARY OF EXPERIMENTS

Data Preparation for Hadoop Streaming is nearly 1.3 times faster than MARISSA at 256 million input records as the former creates the splits more efficiently. This performance gap is expected to grow as the number of records rises from hundreds of millions to billions.

- The Data Transformation (Section III-A2) we propose within our MapReduce streaming pipeline allows conversion of the datasets exported from Cassandra in JSON files to a user specified format. This transformation can be executed without any re-implementation by any MapReduce framework that allows use of non-Java executables.
- Based on the expected data format, the Data Transformation stage can lead to great reduction in data size. This reduction in data size helps the performance of stage MR2 especially for MARISSA whose performance advantage is more visible in smaller data sizes.
- Data Transformation allows users to take a vertical subset of the input database in case processing is only needed to be performed on certain columns.
- The Data Transformation stage under growing input size is nearly 20 percent faster with MARISSA than Hadoop Streaming.
- Data Processing for read intensive applications with MARISSA is nearly 1.5 times faster than Hadoop Streaming up to 256 million records. • When only the Data Processing stage is considered, MARISSA and Hadoop Streaming are over ten times faster than Hadoop-C* under read intensive workloads. However, when the cost of initial steps required for the former two are added, Hadoop-C* becomes nearly three times faster even with the considerable overhead introduced by database reads.
- Under write intensive workloads, MARISSA has a considerable advantage over Hadoop Streaming as HDFS struggles under heavy write load.
- Hadoop-C*-FS offers the best performance under write intensive workloads as it eliminates the cost of data movement and pre-processing by reading the records directly from Cassandra and writing the output to the file system to exclude the overhead of database writes. • In case the data updates are not a concern and/or there are various Data Processing operations to be performed, it is better to take a snapshot of the database to eliminate the overhead of database operations on each run. For such cases, we recommend that Data Preparation and Data Transformation phases should be run just once to prepare the input for the applications that are to be executed in the Data Processing stage. • For process-intensive applications, where the cost of Data Preparation and Data Transformation is significant especially for large data sizes, Hadoop-C* is an ideal option as it almost three times faster than the MapReduce streaming alternatives. • For increasing cluster sizes, even data distribution through worker nodes expectedly provides significant

speedup at Data Preparation and Data Transformation stages. Additionally, increased mapping capacity allows Data Transformation and Data Processing to perform faster processing of input records created by the previous stages.

VI. RELATED WORK

There are various examples of using NoSQL technologies with MapReduce. DataStax offers a "Big Data" system built on top of Cassandra which also supports Apache Hadoop, Hive and Pig. Hive is an open source project built on top of Hadoop to offer querying support over the datasets residing in distributed file systems like HDFS. By contributing read only extensions to the Project Voldemort. Sumbaly et al.aim to provide better batch computing performance when used with Hadoop. Silbertein et al., on the other hand, enhance the bulk insertions of PNUTS in order to improve performance with batched workloads. Sun et al. show processing fast growing RDF datasets, indexed and stored in an HBase cluster, with Hadoop. Ball et al. present Data Aggregation System to collect and query relational and non-relational datasets through a single interface. DAS uses MongoDB for various caching and logging operations. Taylor et al.combine Hadoop and HBase for Bioinformatics to provide a scalable data management and processing platform. They show examples of running Bioinformatics applications like BLAST with Hadoop Streaming but do not provide a detailed study of cases when the target data is stored in the distributed database. OConnor et al. propose SeqWare Query Engine, to provide a querying platform for genome data. They use HBase as a backend storage and use a web query interface to allow access to the datasets. They use the MapReduce model on such platforms and provide a scalable storage, querying and processing framework. Twister is an iterative MapReduce platform whose iterative nature makes it a promising candidate for the streaming pipeline we proposed in Section III-
A. However, it does not support non-Java executables and is out of the scope of this paper. Kaldewey et al.introduce Clydesdale for processing structured data with Hadoop. They provide a comparison study versus Hive showing the performance advantages and argue that MapReduce, more specifically Hadoop, in fact is a compelling platform for structured data analysis but unlike us they do not use datasets from NoSQL databases using non-Java applications. Abouzeid et al propose a hybrid platform combining PostgreSQL and Hadoop to achieve performance similar to the parallel database systems while exploiting fault-tolerance, scalability and flexibility abilities of Hadoop MapReduce. While HadoopDB offers high flexibility by merging Hadoop and DBMS without any code modification, they do not provide alternatives for using non-Java applications in such a setting. Yang et al. with Osprey introduce a middleware to provide MapReduce like fault tolerance support to SQL databases. In classic MapReduce style, Osprey splits the queries into sub-queries and distributes the replicated data throughout the cluster. They provide load balancing and fault tolerance support to the SQL database but do not focus on data processing on such systems using the MapReduce model.

VII. CONCLUSION

In order to fully exploit "Big Data" sets, we need a software pipeline that can effectively combine the use of data stores such as Cassandra with scalable distributed programming models such as MapReduce. In this paper we show two different approaches, one working with the distributed Cassandra cluster directly to perform MapReduce operations and the other exporting the dataset from the database servers to the file system for further processing. We introduce a MapReduce streaming pipeline for the latter approach and use two different MapReduce streaming frameworks, Hadoop Streaming and MARISSA, to show the applicability of our system under different platforms. Furthermore, we present a detailed performance comparison of each approach under various application scenarios. Our results are summarized in Section V to help users make informed decisions for processing large Cassandra datasets with MapReduce using non-Java executables.

REFERENCES

1. Apache Hadoop. http://hadoop.apache.org.
2. Apache HBase. http://hbase.apache.org.
3. http://wiki.apache.org/cassandra/Operations.
4. Datastax. http://www.datastax.com/.
5. National Energy Research Scientific Computing Center. http://www.nersc.gov.
6. Project voldemort. http://www.project-voldemort.com/voldemort/.
7. A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. Proceedings of the VLDB Endowment, 2(1):922–933, 2009.
8. G. Ball, V. Kuznetsov, D. Evans, and S. Metson. Data aggregation system-a system for information retrieval on demand over relational and non-relational distributed data sources. In Journal of Physics: Conference Series, volume 331, page 042029. IOP Publishing, 2011.
9. B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. Proceedings of the VLDB Endowment, 1(2):1277–1288, 2008.
10. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008.
11. E. Dede, Z. Fadika, J. Hartog, M. Govindaraju, L. Ramakrishnan, D. Gunter, and R. Canon. Marissa: Mapreduce implementation for streaming science applications. In E-Science (e-Science), 2012 IEEE 8th International Conference on, pages 1–8, 2012.
12. E. Dede, B. Sendir, P. Kuzlu, J. Hartog, and M. Govindaraju. An evaluation of cassandra for hadoop. In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD '13, pages 494–501, Washington, DC, USA, 2013. IEEE Computer Society.

13. E. Dede, B. Sendir, P. Kuzlu, J. Weachock, M. Govindaraju, and L. Ramakrishnan. A processing pipeline for cassandra datasets based on hadoop streaming. In Proceedings of the 2013 IEEE Big Data 2014 Conference, Research Track, BigData '14, Anchorage, AL, USA, 2014.
14. J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In HPDC, pages 810–818, 2010.
15. Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan. MARIANE: MApReduce Implementation Adapted for HPC Environments. Grid 2011: 12th IEEE/ACM International Conference on Grid Computing, 0:1–8, 2011.
16. Z. Fadika, E. Dede, J. Hartog, and M. Govindaraju. MARLA: MapReduce for Heterogeneous Clusters. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), CCGRID '12, pages 49–56, Washington, DC, USA, 2012. IEEE Computer Society.
17. Z. Fadika and M. Govindaraju. LEMO-MR: Low Overhead and Elastic MapReduce Implementation Optimized for Memory and CPUIntensive Applications. Cloud Computing Technology and Science, IEEE International Conference on, 0:1–8, 2010.
18. Z. Fadika, M. Govindaraju, R. Canon, and L. Ramakrishnan. Evaluating Hadoop for Data-Intensive Scientific Operations. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 67–74. IEEE, 2012.
19. Fadika, Zacharia and Govindaraju,Madhusudhan and Canon, Shane and Ramakrishnan,Lavanya. Evaluting hadoop for data-intensive scientific operations. IEEE Cloud Computing, 2012.
20. T. Kaldewey, E. J. Shekita, and S. Tata. Clydesdale: Structured data processing on mapreduce. In Proceedings of the 15th International Conference on Extending Database Technology, EDBT '12, pages 15– 25, New York, NY, USA, 2012. ACM.

DOI 10.17148/IJARCCE.2017.61106