Evaluating Hadoop for Data-Intensive Scientific Operations

Zacharia Fadika, Madhusudhan Govindaraju
SUNY Binghamton
Binghamton, NY 13902
Email: {zfadika, mgovinda}@cs.binghamton.edu

Richard Canon, Lavanya Ramakrishnan
Lawrence Berkeley National Lab
Berkeley, CA 94720
Email: {scanon, lramakrishnan}@lbl.gov

Abstract—Emerging sensor networks, more capable instruments, and ever increasing simulation scales are generating data at a rate that exceeds our ability to effectively manage, curate, analyze, and share it. Data-intensive computing is expected to revolutionize the next-generation software stack. Hadoop, an open source implementation of the MapReduce model provides a way for large data volumes to be seamlessly processed through use of large commodity computers. The inherent parallelization, synchronization and fault-tolerance the model offers makes it ideal for highly-parallel data-intensive applications. MapReduce and Hadoop have traditionally been used for web data processing and only recently been used for scientific applications. There is a limited understanding on the performance characteristics that scientific data-intensive applications can obtain from MapReduce and Hadoop. Thus, it is important to evaluate Hadoop specifically for data-intensive scientific operations - filter, merge and reorder - to understand its various design considerations and performance trade-offs. In this paper, we evaluate Hadoop for these data operations in the context of High Performance Computing (HPC) environments to understand the impact of the file system, network and programming modes on performance.

I. INTRODUCTION

Scientists are increasingly handling a deluge of data in various disciplines including bioinformatics, climate science, astrophysics [3]. Emerging sensor networks, more capable instruments, and ever increasing simulation scales are generating data at a rate that exceeds our ability to effectively manage, curate, analyze, and share it. The data-deluge experienced by scientific domains is fundamentally changing the scientific process and the associated software stack.

Google proposed the MapReduce model [8] to address the problem related to processing large volumes of Internet data. MapReduce is inspired from functional programming, and has a "map" function, provided by the user, uniformly applied by a cluster of computers to locally stored input blocks. Following the map function, a user defined "reduce" function is subsequently applied to the map intermediary output, in order to consolidate it into a final output. The model takes into account data locality, i.e., move the computation closer to the data and provides mechanisms for fault tolerance to manage the inherent failure rates in commodity computers. Apache Hadoop is an open source MapReduce implementation that has gained significant traction in the last few years in the commercial sector. In recent times, there have been some efforts to port scientific applications to Hadoop [22], [9], [15]. There is a limited understanding of the type of data-intensive operations in scientific workloads that Hadoop might be most suitable for and the various trade-offs associated with programming modes and system characteristics. Specifically, in this paper, we evaluate Hadoop using synthetic benchmarks that represent the categories of data analysis operations, to understand the impact of the file system, network and programming model on performance. Specifically, we make the following contributions:

- We evaluate and determine the suitability of MapReduce/Hadoop for three data intensive operations - filter, merge and reorder.
- We evaluate the effect of system characteristics such as file system and networking.

The rest of this paper is organized as follow. We discuss related work in Section II. We provide background on our evaluation criteria and discuss our methodology for evaluation in Section III. We present our results and discussion in Section IV. Conclusions and future directions are presented in Section V.

II. RELATED WORK

Several groups have looked at Hadoop for science, developed benchmarks for studying Hadoop characteristics. However there has been limited prior work that evaluates Hadoop for scientific data-intensive workloads at HPC centers. Here we summarize related work to our research.

MapReduce comparisons and Hadoop benchmarks. Stonebraker et. al. [25] compare MapReduce and Parallel databases. Previous work has analyzed whether high-performance interconnects can help Hadoop applications [26]. Gu et. al [11] compare and contrast Hadoop and Sphere and discuss some of the design trade-offs necessary in data intensive computing. Starfish [12] uses self-tuning database principles to provide auto-tuning for Hadoop's configuration parameters.

Kontagora et. al. [17] benchmark Hadoop’s performance in virtualization, and quantifies the difference observed in contrast to a non-virtualized environment. HiBench is a set of micro and synthetic benchmarks for Hadoop MapReduce [13]. HiBench is however geared towards traditional MapReduce applications rather than scientific applications.

Jiang et. al. [14] offer an in-depth study of MapReduce performance for traditional MapReduce applications. MRBench
[16] is a benchmarking tool implemented for comparing Hadoop to relational database systems. MRBench is based on TPC-H which is used to evaluate database systems that have realistically complex queries. TPC-H queries are converted into MapReduce jobs and each job includes multiple steps of MapReduce. The output produced by each step becomes the input for the next step except for the last step which collects the result of the operation. MapReduce/Hadoop benchmarks have been used in the design of Gordon, a system architecture at San Diego Supercomputing Center for data-centric applications combining low-power processors, flash memory.

In our previous effort [10], we focused on benchmarking usage scenarios such as memory and CPU-intensive applications in different MapReduce implementations. The current work focuses specifically on benchmarking data-intensive operations in Hadoop running on HPC systems.

**HDFS and Other File Systems.** The portability vs performance trade-offs of HDFS [23] has been explored previously. However, there is a limited understanding of applications that might perform with other file systems with Hadoop's job framework. Previously [5], the performance of a traditional cluster file system and a specialized Internet file system for a variety of workloads has been compared and it is projected that many of the features required for cloud workloads can be built on traditional POSIX-based cluster file systems.

**Hadoop and Science.** CGL-MapReduce is a set of MapReduce APIs for use with Hadoop MapReduce in processing a set of binary data formats in use with High Energy Physics [9]. Previously, it has been shown that MapReduce can be useful for a myriad of disciplines, ranging from bioinformatics to astrophysics, to Cybersecurity with appropriate extensions such as arbitrary input format support required for scientific applications [18], [28]. Buck et al. [6] introduced a Hadoop plugin allowing scientists to specify logical queries over array-based NetCDF data models. Nguyen et al. [19] recommend the use of MapReduce for climate satellite applications. The effects of data organization on MapReduce performance has been shown [27], however the performance of the model for different data-intensive scientific operations has not been studied.

### III. Methodology

Early studies have shown that Hadoop can be useful to scientific applications [20], [21], but numerous challenges (e.g., lack of POSIX support, dependence on Java) for its use with scientific applications have been noted. There is limited insight into the performance impact of some of the design trade-offs that are entailed when using Hadoop for data-intensive operations. In this section, we provide a brief overview of our methodology.

**A. Background**

The MapReduce model and Hadoop were designed to be implemented and deployed on very large clusters of commodity machines. Hadoop consists of two core components: the job management framework and the Hadoop Distributed File System (HDFS) [24] and they typically rely on a 1GigE backend.

Hadoop Distributed File System (HDFS) differs from parallel file systems due to the underlying storage model. HDFS relies on local storage on each node while parallel file systems are typically served from a set of dedicated I/O servers. Thus, HDFS is able to serve a larger bandwidth for concurrent access than most high performance file systems. Most parallel file systems use the POSIX interface that enables applications to be portable across different file systems on various HPC systems. However, HDFS is non-POSIX compliant making it hard for applications that already have legacy codes or need the flexibility and portability of its code across platforms. Scientific users heavily rely on POSIX compliant file systems at HPC centers for storing and accessing their data produced from their experiments and simulations. In this paper, we evaluate the performance implications of using Hadoop's core job management framework with POSIX high performance file systems available at HPC centers.

MapReduce and Hadoop carefully consider data locality in the map phase, there is minimal data movement early in the job flow. However, the shuffle phase sorts all the key-value pairs from the maps and assigns a set of keys to the reducers. This results in large data movements across the cluster. A number of HPC machines where today's scientific applications run have high performance networks such as Infiniband readily available. We evaluate and analyze the impact of the network for scientific applications using Hadoop (without any changes to the internals).

Apache Hadoop and a number of the other open source MapReduce implementations are in Java and by default expect the map and reduce functions to be Java programs. Scientific codes are often written in Fortran, C, C++ or use languages such as Python for analysis. The Hadoop streaming model allows one to create map and reduce jobs with any executable or script as the mapper and/or the reducer. This is the most suitable model for a majority of scientific applications. The streaming mode allows scientific applications to easily scale up using the MapReduce framework while using existing codebases. However, there is still a limited understanding of this might impact performance.

**B. Data Operations**

Data-intensive scientific applications are relatively new and there is little in the way of representative application benchmarks that can be used to evaluate solutions. Developing a comprehensive benchmarking and associated understanding of classes of applications is outside the scope of this paper. However, we attempt to capture three high-level data intensive operation categories based on I/O pattern in our evaluation and study their effects in a Hadoop environment.

Similarly, very little is known about the effect of granularity of jobs for data-intensive sciences. For example, it is unclear if it is better to have more tasks processing on smaller amounts of data or a smaller set of tasks where each task processes
large amounts of data. This depends on the overheads of the system and application characteristics.

Based on our experience with scientific use cases, we identified three common data operations:

**Filter.** A Filter operation is when data is analyzed and the output result is a subset of the entire data set. Scientific applications that involve searching for a certain pattern would fit this kind of data operation. The volume of input data processed is significantly larger than the volume of output data. For example, Filter applications are common in Geographic Information Systems. In the case of spatial data processing, a large amount of aerial imaging data is indexed to provide better satellite image resolution [7]. While the raw data processed is extremely large, the processing filters the data by removing redundant and undesired areas, to produce a smaller, higher resolution and more focused satellite image.

**Reorder.** A Reorder operation is when the input is reordered in some way resulting in an output dataset that is close to identical in size to the input. Sorting the input data is an example of this kind of operation. For example, in [2] the authors start with a long and fixed gene sequence, subsequently shuffling it to provide a different combination of the same genetic sequence. In this case, the input used exactly equals the output produced.

**Merge.** Some scientific applications result in the merging of two or more data sources. For example, observation data might need to be merged with simulation data. Such Merge operations result in output data volumes, significantly larger than the primary input data size. For example, real world seismic events data needs to be linked to regional ground characteristics [4]. Subsequently, the researchers simulate the impact of a similar event in a different part of the world sharing the same ground characteristics. In their experiments, they generate the ground model for a 600KM long x 300KM wide x 100KM deep region in southern California where the simulation produces large output data from smaller sample input data.

C. Workloads

Data-intensive science applications have varying characteristics. The increasing volume, complexity and the velocity (i.e., rate of data change) of data are considered key characteristics of data-intensive applications. In order to exercise control over the experiments, we consider three different workloads and data sets with increasing degrees of data volume. The amount of data we process in this paper is similar in size to the large data sets processed with scientific applications today.

We implemented three synthetic benchmarks for our experiments. The actual processing elements in our benchmarks are intentionally simple since our focus is primarily on the data operations. Additionally, in our experiments we measured the input reading, processing and output writing times to highlight where the bottlenecks might lie. We use a simple palindrome processing program to represent a light-weight application that does minimal I/O and processing. We use the census data as representative of mid-sized volume of data and Wikipedia data processing as representative of an application with a large volume of data. All tests were implemented in Java using the Hadoop API unless otherwise mentioned. Section IV focuses on the Wikipedia results due to the volume of the data. Trends from other applications are noted as applicable.

**Palindrome Checking.** Science workloads operating on little or no disk input and producing if not none, very little disk output are considered under the minimal I/O applications category. We use palindrome check as the representative application for this category in our experiments.

Palindrome check performs a palindrome check on numbers given as part of a numeric range and reports as output the number of palindromic occurrences within the given range. We control the inputs and outputs for this application to simulate the three data operations. For example, to represent a Merge all the numbers processed are written to our output, along with the result produced by the experiment.

**Census Data Processing.** The census data is about 300GB. Our Filter Hadoop application extracts the counties with a bigger female population than a male population. Even though the raw data is very large, the resulting output is rather small. To represent the Merge operation, we label the census data for human search and viewing. Labels are simply added to each line of the data, using MapReduce, making this application, far more write intensive than read intensive. Finally, to represent the Reorder operation, we simply replace the """" separator throughout the entire census data with """:"

**Wikipedia Data Processing.** The Wikipedia data is 6TB. We use subsets of the Wikipedia data for smaller size experiments as noted in the results section (Section IV). We developed an application that does page title indexing on the Wikipedia data. The bulk of the Wikipedia data is composed of raw text and numeric values. Each article has a title identifying a single topic that may span multiple pages. In searching such an immense archive, it would be helpful to provide an index of all existing articles in the archive.

For the Merge operation, we index each line of our 6TB data set. This operation is data intensive as 43% more data compared to the input is written back. Finally, for the Reorder operation, we perform a data conversion operation on our Wikipedia data. Assuming that our search framework recognizes the Wikipedia timestamp tag as \(<\text{date-time}>\) and \(<\text{date-time}>\), rather than \(<\text{timestamp}>\) and \(<\text{timestamp}>\), we need to transform our data for compatibility with such a system. The amount of data read is the same being written. Even \(<\text{timestamp}>\) and \(<\text{date-time}>\) contain the same amount of characters.

**TeraGen.** In addition to the above workloads, we use TeraGen from the Hadoop distribution. TeraGen is a write-intensive operation and writes out a specified amount of data.

D. Machines

All experiments were conducted on the Magellan NERSC Hadoop testbed. Magellan is a IBM iDataPlex cluster. Each node has two quad-core Intel Nehalem processors running at 2.67 GHz, 24 GB of RAM and two network connections:
a single Quad Data Rate (QDR) Infiniband (IB) network connection and a GiB ethernet connector. The IB network is locally a fat-tree with a global 2D-mesh. There were 77 nodes in the Hadoop cluster at the time the experiments were performed. The Hadoop version on the testbed was Cloudera version 0.20.2 + 228. Additionally, the high-speed NERSC GPFS file-system was used. This file system uses IBM’s GPFS and has a peak performance of approximately 15 GB/sec.

Fig. 1. Palindromic processing for 0.1TB of data showing read, write and processing times for Filter, Merge and Reorder. The Filter operation is significantly faster than other operations since it does negligible I/O.

Fig. 2. Palindromic processing for 1TB of data showing read, write and processing times for Filter, Merge and Reorder. Filter is the fastest since it has minimal writes. In this case the Reorder does less writes than Merge and hence is faster. Merge takes 42x longer than Filter.

IV. RESULTS

In this section, we outline our performance results to understand the effects of the data operations in HPC environments. All experiments were repeated three times.

A. Understanding overhead of operations

Figure 1 shows Palindromic check for 0.1TB of data for Filter, Merge and Reorder operations. Additionally, Figure 1 details time slices pertaining to read, write and processing time for each application class. As described in section III-C, the Filter application reads more than it writes, while Merge writes more data compared to the amount it reads. As Filter applications do not require much writing, it is the cheapest operation followed by the Merge and the Reorder operations. The runtime incurred by Merge and Reorder (Figure 1) is dominated by their write times, highlighting the costly nature of write intensive applications over their read and processing intensive counterparts.

Figure 2 shows Wikipedia data processing with 1TB of data. As this graph shows, the trends first observed in Figure 1 hold true in this experiment. Given the differences in data and processing however, the Reorder operation here consumes less time than its Merge counterpart as its algorithm undertakes less write operations in this context than Merge does. Merge takes 42x longer than Filter even for 1TB of data due to amount of writes. Thus, writes are relatively more expensive which might also be attributed to the fact that each file written out is replicated twice more by Hadoop. Additionally, Hadoop uses data locality for reads in the map phase and thus read operations are typically less expensive. This trend holds across different applications, different data types and over different data sizes.

B. Effects of number of maps

Figure 3 depicts a Filter Wikipedia data processing operating on 1.1 TB input hosted on HDFS and GPFS. This experiment shows the performance of both file systems when dealing with read intensive applications. As the number of mappers is gradually increased, a performance improvement is observed. GPFS initially shows much better performance than HDFS as the number of total mappers and thus parallel reads is kept at a minimum. As parallel reads are increased with the addition of more mappers, GPFS performance slowly degrades even as it still performs better than the HDFS at around 8000 mappers. It is important to note here that our map capacity on the cluster was 462 (77 nodes x 6 maps/node). Thus, only
462 mappers run concurrently but as the load increases with the mappers and more writes, we see HDFS is able to sustain better I/O bandwidth through disks attached to the local nodes. The result is as expected since HDFS is expected to perform better than other systems when running at larger scales.

C. Effects of file system

![Graph showing performance comparison between HDFS and GPFS](image)

Fig. 4. Comparison of HDFS and GPFS for Filter, Merge and Reorder with read, write and processing time slices on 2 TB. At given scales, GPFS shows better performance than HDFS For Merge and Reorder, Filter shows comparable performance on GPFS and HDFS.

Figure 4 shows the comparison of HDFS and GPFS for all three data operations for a 2TB data set of Wikipedia. The figure shows the split of the processing time. The difference between HDFS and GPFS is negligible for the Filter operation. We notice that for the Reorder and Merge operations, GPFS achieves better performance than HDFS overall at this data size. However, HDFS performs better than GPFS at reads due to data locality whereas GPFS performs better than HDFS for the write part of the application at the given concurrency.

Figure 5 shows the comparison of performance of the data operations on both GPFS and HDFS for Wikipedia for varying input sizes. For the Filter operation, there is negligible difference between HDFS and GPFS performance until 2TB. However at 3TB, HDFS performs significantly better than GPFS. For the Reorder and Merge, GPFS seems to achieve better performance than HDFS and the gap increases with increasing file sizes.

Figures 6 show the comparison of using HDFS and GPFS as the underlying file system for TeraGen with varying number of maps for Wikipedia. HDFS and GPFS have been designed for different usage scenarios and the goal of our comparison is not a quantitative performance comparison of the two file systems. Each of these file systems has its own strengths for certain workloads. Our goal here is to understand if scientific applications can benefit from Hadoop’s job management framework while using POSIX compliant file systems available in HPC centers.

Figure 6 a) shows the time for TeraGen to generate 1 TB of data on both file systems. We see that the performance of GPFS shows a slight decrease as the number of concurrent maps are increased. On the other hand, HDFS’s performance significantly improves as number of maps increases as HDFS is able to leverage the additional bandwidth available from disks on every compute node. Figure 6 b) shows the effective bandwidth for both systems and we can see that HDFS’s effective bandwidth is steady and increasing. Hadoop and HDFS have been designed to manage high-levels of parallelism for data-parallel applications. These results show that for small to medium scale parallelism, applications can use Hadoop with GPFS without any loss in performance.

D. Effects of network

Science applications running in HPC centers traditionally use high performance, low-latency networks. However Hadoop has traditionally been run on commodity clusters based on Ethernet networks. The shuffle phase between the map and reduce phase is considered to be the most network intensive operation since all the keys are sorted and data belonging to a single key is sent to the same reducer resulting in large data movement across the network. Figure 7 shows the comparison of the network on the three data operations with varying file sizes for Wikipedia data set. We observe that Filter and Reorder are not affected much by the changes in the network. The Merge operation shows that the application performs better on the Infiniband network at larger file sizes (about 1% better at 3.0 TB). This is likely due to the growth in the data in the Merge compared with the Reorder or Filter. It is important to note that we did not change the Hadoop framework to efficiently use the high-speed Infiniband network. Other work [1] has documented that the Shuffle algorithm needs to be modified to fully exploit the high-speed network.

E. Effects of replication/data locality

Replication in Hadoop is used for fault-tolerance and increasing the effect of data locality of the input files. Figure 8 shows the effects of varying replication on the Filter and Reorder data operations with a 2TB data set wikipedia data set. For the read-intensive Filter operation increasing the replication factor significantly impacts performance. The Reorder operation benefits from the replication but the effects are minimal due to dominance of write costs.

F. Streaming

Existing scientific applications can benefit from the MapReduce framework using the streaming model. Thus, we constructed the Filter operation in both C and Java and compared the two to understand the overheads of streaming. Comparing the overheads of streaming are tricky since there are some differences in timing induced from just language choices. Hadoop streaming does not support Java programs since Java programs can directly use the Hadoop API. Figure 9a shows the comparison of the timings of both programs running on a single node. We see that the C implementation is more efficient (20% at 0.1TB and 21% at 0.2TB) and also the
Fig. 5. Wikipedia data processing comparison of HDFS and GPFs for varying file sizes for Filter, Record and Merge. For up to 2TB there is negligible difference between HDFS and GPFs but HDFS performs better at 5TB and beyond. GPFs seems to perform better than HDFS for the write intensive Filter and Merge operations.

Fig. 6. Wikipedia data processing on HDFS and GPFs comparison using Teragen at Time by Bandwidth. At smaller concurrent number of mappers, GPFs does outperform HDFS but as number of mappers increases GPFs performance decreases and HDFS is expected to do much better at larger concurrent mappers as expected.

Fig. 7. Wikipedia data processing comparison at Ethernet and Infiniband for various file sizes for a) filter b) aggregate and c) merge applications. The network has minimal effect within the scope of the experiment. However, it is likely HadBag's Shuttle algorithm might need to be modified to see significant benefits.
performance improves as the data size increases. Figure 9b shows the comparison of the streaming C version with the native Hadoop version for varying file sizes. We notice that the performance through Hadoop is similar (within 3 to 5%) for smaller file sizes. This actually indicates that the streaming has additional overhead since the C version is more efficient as shown in Figure 9. As the file size increases we see that the overhead for streaming increases and the streaming C version is almost 20% slower at 1.6TB.

G. Summary

We performed a thorough evaluation of Hadoop in an HPC environment environment. Our results provide some key insights for using Hadoop and design of future software frameworks for data-intensive sciences.

- For the same quantum of work performed on an equivalent data volume, Filter is the least expensive operation due to minimal writes whereas Merge and Reorder are expensive due to the output writes.
- High performance file systems (e.g., GPFS in our case) does better with writes at lower concurrency but HDFS does significantly better at higher concurrencies. HDFS does better for read-intensive applications for larger data volumes. The ability to use existing file systems available at HPC centers provides a number of advantages to scientific applications. It allows users to apply MapReduce functions to existing data, use legacy application binaries and operate through familiar HPC environments.
- There is minimal impact seen due to high-performance low latency interconnects on our benchmarks. To truly leverage, the underlying network the underlying shuffle algorithm of Hadoop will need to be changed. Mellanox recently announced an unstructured data accelerator (UDA) software plugin that will allow Hadoop frameworks to leverage RDMA (Remote Direct Memory Access) [1] and modified the shuffle algorithm to fully
exploit the high-speed network.

- Replication and as a result data locality can improve the performance for read-intensive applications (e.g., Filter). For operations such as Merge and Reorder, the majority of the time is spent in writing out the output and the advantage of replication could be insignificant.

- Hadoop’s streaming mode allows legacy applications to be plugged in as maps and reduces. The streaming mode adds an overhead that increases data sizes compared to using the Java native Hadoop interface.

V. CONCLUSION

MapReduce and Hadoop are expected to be useful to data-intensive scientific applications. However, there is limited understanding of the trade-offs of different design choices in Hadoop applications such as file systems, streaming mode, networking, replication etc. In this paper, we conduct a rigorous evaluation to understand the performance characteristics of typical scientific data-intensive operations in Hadoop. Our study has important implications for MapReduce framework and scientific application developers. MapReduce framework developers must consider the needs of scientific applications including high speed access to legacy data sets stored and access to high speed networks available at HPC centers.

The performance an application can obtain is largely workload dependent, but there are some key factors. First, while streaming mode is a useful mode for legacy scientific applications, it does introduce some overheads that increase as the data size increases. Second, small to mid-range applications with legacy data sets on high performance file systems could use MapReduce as a job management framework with minimal or no loss in performance. Finally, to truly benefit from Infini-, the underlying Hadoop framework’s shuffle algorithm will likely need to be rewritten.

REFERENCES