Benchmarking and Performance studies of MapReduce / Hadoop Framework on Blue Waters Supercomputer

Manisha Gajbe[1], **Kalyana Chadalavada**[1], **Gregory Bauer**[1], **William Kramer**[1,2]
[1]National Center for Supercomputing Applications, University Of Illinois at Urbana-Champaign, IL, USA
[2]University Of Illinois at Urbana-Champaign, IL, USA
$manisha, kalyan, gbauer, wtkramer$@illinois.edu

Abstract— MapReduce is an emerging and widely used programming model for large-scale data parallel applications that require to process large amount of raw data. There are several implementations of MapReduce framework, among which Apache Hadoop is the most commonly used and open source implementaion. These frameworks are rarely deployed on supercomputers as massive as Blue Waters. We want to evaluate how such massive HPC resource can help solving large-scale data analytics, data-mining problems using MapReduce / Hadoop framework.

In this paper we present our studies and detailed performance analysis of MapReduce / Hadoop framework on Blue Waters Supercomputer. We have used standard popular MapReduce benchmark suite that represents wide range of MapReduce applications with various computation and data densities. Also, we are planning to use Intel HiBench Haddop Benchamrk Suite in future. We identify few factors that significantly affect the performance of MapReduce / Hadoop and shed light on few alternatives that can improve the overall performance of MapReduce techniques on the system.

The results we have obtained strengthen our belief in possibility of using massive specialized superrcomputers to tackle big data problems. We demonstrate the initial performance of the MapReduce / Hadoop framework with encoraging results and we are confident that the massive traditional High Perfromance Computaing resource can be useful in tackling the big-data research challneges and in solving large-scale data analytics, data-mining problems.

Keywords: MapReduce, Hadoop, Blue Waters

1. Introduction

MapReduce [1] is a well-known programming framework pioneered by Google for data intensive and large-scale data analysis applications. The architecture is simple abstraction that allows programmers to use a functional programming style to create a *map* function that processes a key-value pair associated with the input data to generate a set of intermediate key-value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. The MapReduce programming model is divided into 3 simple phases namely: *Map; Shuffle and Sort; Reduce* as shown in figure 1.

Map Phase: In the map phase, the input data is partitioned into input splits and assigned to Map tasks associated with processing nodes in the cluster. The Map task typically executes on the same node containing its assigned partition of data in the cluster. These Map tasks perform user-specified computations on each input key-value pair from the partition of input data assigned to the task, and generate a set of intermediate results for each key.

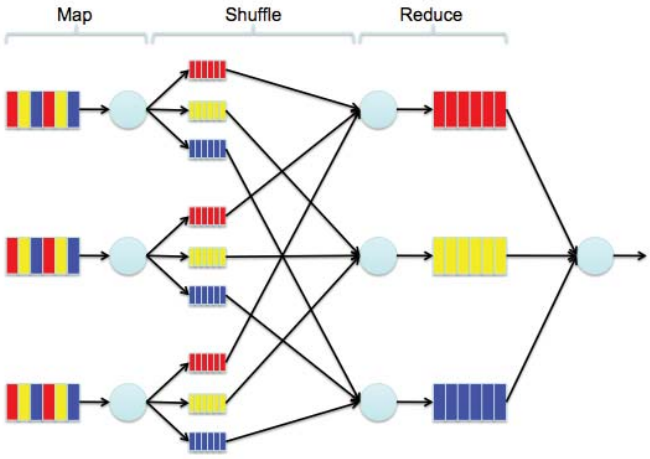

Fig. 1: MapReduce - 3 Phases

The shuffle and sort phase: The shuffle and sort phase sorts the intermediate data generated by each Map task from other nodes and divides this data into regions to be processed by the reduce tasks. This phase also distributes this data as needed to nodes where the Reduce tasks will execute.

Reduce Phase: In reduce phase, the data divided by shuffle and sort phase is processed. The Reduce tasks perform additional user-specified operations on the intermediate data possibly merging values associated with a key to a smaller set of values to produce the output data.

All Map tasks must complete prior to the shuffle and sort and reduce phases. The number of Reduce tasks does not need to be the same as the number of Map tasks. For more complex data processing procedures, multiple MapReduce calls may be linked together in sequence.

The MapReduce programming model is also becoming popular in scientific computing, where scientists need to frequently analyze a large volume of experimental and

simulation data. Such data analysis is often implemented as independent tasks that can be expressed as mapping operations in MapReduce. For example, in genome sequencing, the matching of large number of sequences against a huge collection of known sequences can be considered as mapping of similarity function to pair of sequences. Similarly, for the post-processing of simulation data, the tasks can be expressed as MapReduce, where a single program is run multiple times with different input parameters. MapReduce is a simple and scalable approach that enables scientists to achieve simulation results from large-scale data.

Google implemented MapReduce to execute very large matrix-vector multiplications needed for the PageRank calculations. Matrix operations such as matrix-vector and matrix-matrix calculations fit very well into the MapReduce style of computing. Another examples in numerical computing that can be solved are Singular Value Decomposition or Sparse Matrix Vector Multiplication that are used in lots of HPC applications. Given the large-scale problem size and types that are addressed using MapReduce, and the popularity of MapReduce as an implementation paradigm, it is unquestionable to explore its use on traditional HPC platforms.

In this paper, we conduct initial benchmarking and performance results of MapReduce framework on the Blue Waters [3] petascale system. We have briefly described the challenges of using Mapreduce / Hadoop framework on High Performance Computing (HPC) platforms. We have used Apache Hadoop [2], the most popular and commonly used MapReduce framework. However, there is no official / formal support for Hadoop or related stack on the Blue Waters system.

The rest of the paper is organized as follows: Section 2 discusses the challenges while deploying and using MapReduce / Hadoop on a High Perfoprmance Computing resource. In Section 3, we describe the architecture of computing systems used and brief description on the test cases and benchmarks we have experimented with. We talk about experimental setup along with benchmarking environment and Hadoop related configuration that we used and challenges faced in deploying Apache Hadoop software stack on the Blue Waters system in Section 4 follwed by results with discusion in Section 5 . Related work is briefly reviewed in Section 6 and finally we conclude and discuss future work in Section 7.

2. Challenges on HPC System

There are certain challenges on how MapReduce-Hadoop framework will fit into HPC environment. We have come across few of them while working with the MapReduce programing paradigm on the Blue Waters system.

2.1 Parallelism:

Most of the HPC applications use divide-conqure method and each task communicate with other tasks frequently. These applications are often classified according to how often their subtasks need to synchronize or communicate with each other. In applications that exhibit fine-grained parallelism the subtasks communicate frequently while applications with coarse-grained parallelism the subtasks do not communicate many times. Other types of HPC applications are embarrassingly parallel that rarely or never have to communicate. Embarrassingly parallel applications are considered the easiest to parallelize. MapReduce completely relies on Embarrassingly Parallel (EP) techniques. May of the HPC applications do not fall in this category. Programmers will need to rewrite the codes to expose the EP method in their existing codes. Also the programming models such as MPI, OpenMP, OpenACC etc developed for Parallel Programming are not suitable for MapReduce / Hadoop framework.

2.2 Programming Language:

The main programming language for HPC applications is either Fortran or C while Hadoop is written in Java so that the code written can run on any hardware platform. This is completely opposite when it comes to traditional HPC applications, where they are compiled and optimized for specific software and hardware platform on which they will run. As per the HPC users, the codes written in Java are slow and inefficient which is not acceptable in HPC coomunity. Another reason is, Hadoop was essentially designed for world wide web services, for which Java is almost perfect language of programming, while HPC applications address wide range of scientific applications that are developed historically.

2.3 File System:

The main requirement of Hadoop is availability of local data storage. However, for the HPC systems there is no local storage. The file system is shared across all the available nodes. Most of the time the file system is either General Parallel File System (GPFS) [32], [33] or Lustre [34]. Simulating these shared files system as a local storage is not straight forward. Additionally, these filesystems extensively use POSIX, while Hadoop doesn't support it.

2.4 Resource Management:

In traditional hadoop clusters the resource management is entirely handled by the hadoop framework and the types of workloads are similar. On the other hand, typical HPC systems handle various types of workload and the resource management is taken care by dedicated scheduling software such as Moab [35], PBS or Slurm. Integrating these resource scheduler with Hadoop is not a simple task.

2.5 Operating System:

Hadoop framework requires a full flavored operating system. Current HPC systems have stripped down version of linux kernel to reduce unwanted polling from the Operating System (OS) which in turn improves the performance of an application on the system. To use the MapReduce / Hadoop framework one will have to use the Cluster Compatibility Mode on the given system so that a full flavor of OS is available for MapReduce applications.

2.6 IP stack over interconnect:

Hadoop framework uses TCP / IP or Ethernet and not high speed and lossless Remote direct Memory Access (RDMA) technologies. Hadoop does not support low latency high speed interconnect with scalable topologies like 3D Torus or 5D torus or Dragonfly or Gemini etc. It supports only multi-stage clos style network [38]. The network topologies mentioned above are relevant only to HPC or Supercomputing.

We will have to look into all the above challenges to evaluate the MapReduce / Hadoop framework on Blue Waters system. However, solutions are being developed and attempted to address a few of the above mentioned challenges.

3. System Overview

In the section we describe the architecture of the computing system we have used for the benchmarking and performance evaluation of MapReduce framework and brief description of test cases and benchmarking candidates. We have used our Test and Development System as well as Blue Waters [3] system to perform our experiments.

3.1 Computing System Overview

The hardware we used is the sustained petascale system of Blue Waters [3] hosted at the University of IllinoisâĂŹs National Center for Supercomputing Applications (NCSA) and funded by NSF. Blue Waters is one of the largest computational resources in the world, serving NSF Science community researchers throughout the United States.

3.1.1 JYC Configuration

JYC is our Test and Development System (TDS) where we test and evaluate software and changes before we deploy it on Blue Waters. JYC is a single rack XE6m/XK6m. There are 96 total nodes with a aggregate peak compute performance of ~30.3 TF.

- 76 nodes are XE6 nodes with (1216 bulldozer modules, 2432 integer cores, 313 GF/node, 23.8 TF total):
  - two AMD Interlagos processors (16 Bulldozer modules total)
  - 64 GB of RAM
- 8 nodes are XK6 nodes with (64 bulldozer modules, 128 integer threads, 156 GF x86/node, 655 GF/Fermi, 6.5 TF total):
  - one AMD Interlagos processor (8 Bulldozer modules)
  - 32 GB of RAM
  - one NVIDIA Fermi GPU (with 6GB of RAM)
- The remaining 12 nodes are service nodes used for boot, sdb, LNET routers, login, and network gateway. All 96 nodes are on the gemini interconnect which is cabled as a 2D mesh 1D torus. The login and network gateways each have a dual-port 10Gb Ethernet NIC.
- JYC also has Lustre as underlying file system, Torque as resource manager and Cluster Compatibility mode.

3.1.2 Blue Waters Configuration

Blue Waters is a Cray XE6-XK7 supercomputing system managed by the National Center for Supercomputing Applications for the National Science Foundation. The system has a peak performance of 13.34 PF, aggregate IO throughput in excess of 1 TB/s, 26 PB online disk capacity and nearly 200 PB of nearline storage. Blue Waters contains two types of compute nodes: XE6 and XK7. There are 22,640 XE6 nodes and 4,224 XK7 nodes. Each XE6 node has two 16 core AMD 6276 CPUs, 64 GB of main memory. Each XK7 node has one 16 core AMD 6276 CPU, 32 GB of main memory and one Nvidia Kepler K20X graphics processing unit (GPU) with 6 GB of GDDR5 on-board memory. The compute and file system nodes are interconnected using the Cray Gemini high speed interconnection network. Two nodes share a single Gemini ASIC (Application-Specific Integrated Circuit), which contains two network interface controllers (NICs) and a YARC-2 router. The network is organized in a 24 X 24 X 24 3D torus topology.

3.1.3 File System

The file system on Blue Waters will be built using Cray Sonexion 1600 Lustre appliances. The Cray Sonexion 1600 appliances provide the basic storage building block for the Blue Waters I/O architecture and are referred to as a "Scalable Storage Unit" (SSU). Each SSU is RAID protected and is capable of providing up to 5.35 GB/s of IO performance and approximately 120TB of usable disk space. The *scratch* file system where the runs were made uses 180 (one hundred eighty) SSUs to provide 21.6 PB of usable disk storage and 963 GB/s IO performance. This file system can provide storage for up to 2 million file system objects. This file system is high performance, high capacity transient storage for applications.

3.1.4 Resource Management

The Blue Waters system uses TORQUE Resource Manager [36] integrated with the Moab Workload Manager

to schedule and manage user jobs. Torque is based on OpenPBS, most of the commands for managing your jobs on Blue Waters will be the same as PBS commands. The application launcher (aprun) utility on the Cray system launches applications on compute nodes similar to mpirun on many other systems to launch jobs. Application Level Placement Scheduler (ALPS) take care of job placement and execution of the applications submitted by aprun.

The Blue Waters system also has Cluster Compatibility Mode (CCM), a component of Cray environment to support full Linux compatibility mode. With help of CCM, XE/XK compute node, normally carrying a stripped down operating system, can be turned into a typical node in a standard Linux cluster. This mode is used to run programs on the MapReduce / Hadoop framework.

3.2 Benchmarks Information

In this section we describe the standard and industry benchmarks we have used during our experiments. Some of them are available with the Hadoop distribution while others are developed by the academia or industry.

3.2.1 PI Calculation

PI is a MapReduce program that estimates the value of *PI* using a quasi-Monte Carlo method. This program is available with the Hadoop distribution. *PI* is a purely computational application that employs a Monte Carlo method to estimate the value of *PI*. It is very nearly "embarrassingly parallel": the map tasks are all independent and the single reduce task gathers very little data from the map tasks. There is little network traffic or storage I/O. Detailed information on *PI* program can be found here [22]. BBP is a MapReduce program that uses Bailey-Borwein-Plouffe [39] to compute exact digits of *PI*.

3.2.2 Word Count

Word Count is a MapReduce program that counts the words in the input files. The program counts the occurrences of each word in a large collection of documents. Map emits <word,1> tuples. Reduce adds up the counts for a given word from all map tasks and outputs the final count.

3.2.3 Grep

Grep is a MapReduce program that counts the matches of a regex in the given input. It is helpful in searching a pattern in a file and is a generic search tool used in many data analyses. Each map task outputs lines containing either of the patterns as <regex, 1> tuples. Reduce task adds up the counts and emits <regex, n> tuples. This program is available with the Hadoop distribution.

3.2.4 NNBench

NNBench is a benchmark that stresses the namenode. It is useful for load testing the NameNode hardware and configuration of underlying filesystem. The *NNbench* program is part of Apache Hadoop distribution that can simulate requests for creating, reading, renaming and deleting files on the Hadoop filesystem.

3.2.5 DFSIO

The *DFSIO* program is part of Apache Hadoop distribution to compute the aggregated bandwidh delivered by HDFS. It is a read and write test for the filesystem. The test handles large number of tasks performing read or write operations in parallel. This test run as a MapReduce job, where each map task *i* opens a file to read or write and measures number of bytes transferred and the ececution time for that task. Map tasks followed by a single reduce task for post-processing that aggregates the results from all the map tasks by computing average I/O rate and average throughput for each map task. More information on how to run the benchmark and interpreting the results obtained is explained in [20].

3.2.6 Intel HiBench

Intel's HiBench [4], a Hadoop benchmark suite consisting of both synthetic micro-benchmarks and real world applications such as Sort, WordCount, TeraSort, Bayes, KMeans, NutchIndexing, PageRank, DSFIOE. It can be used as a representative proxy for benchmarking Hadoop applications.

3.2.7 MRBS

MRBS [21] is a comprehensive benchmark suite for evaluating the performance of MapReduce systems. MRBS contains five benchmarks covering several application domains and a wide range of applications that are data-intensive versus compute-intensive or batch applications versus online interactive applications.

4. Experimental Setup

In this section we describe the benchmarking environment along with the configuration setting we used for Hadoop deployment on Blue Waters system and we detail some of the challenges faced in deploying Apache Hadoop software stack on the Blue Waters system.

4.1 Benchmarking Environment

We have used JYC, the Test and Developemnt System (TDS) as well as Blue Waters system for our experiments. Blue Waters has 22640 XE and 4224 XK nodes while JYC consists of 76 XE and 8 XK nodes. The file system is Lustre which is shared across all the nodes.

Resource management and scheduling is handled by Torque. So each job may or may not get different nodes in the systems and at different network location. We have integrated Yarn with the existing resource management and scheduling software. We use *ccmrun* supported by Cluster

Compatibility Mode on Cray systems to properly launch the MapReduce / Hadoop workload on the system.

4.2 MapReduce and Hadoop Settings

We have used an Open Source distribution of Apache Hadoop stack 2.3.0. The node manager resource memory is set to 52 GB which is approximately 80% of memory available on a single compute node. The value for *cpu-cores* is set to 32, virtual core to physical core ratio is set to 2 and virtual memory to physical memory ratio is set to 2. The memory per container is set to 2 GB, therefore we can have 25 number of containers per node. The heap sizes for map task and reduce task are set to 1.6 GB and 3.2 GB respectively. The detailed information on how to set these parameters is available at [27] and we have followed these instructions. We have used same settings on both JYC and Blue Waters systems.

4.2.1 myHadoop

myHadoop [19] is a framework used for configuring Hadoop on traditional HPC resources using the standard job scheduling and resource manager software. User can run Hadoop codes on the HPC resource without having root privileges using myHadoop. It supports a regular non-persistent mode where the local file system on each compute node is used as the data directory for the Hadoop Distributed File System (HDFS), and also a persistent mode where the HDFS can be hosted on a shared file system such as Lustre or GPFS. We have used myHadoop version 2.1.0.

4.2.2 Yarn

MapReduce / Hadoop workloads are excuted on standard Hadoop cluster with the help of resouce management and scheduling entirely handled by Hadoop framework. In contrast, the resouce management and scheduling is always handled by special type of dedicated software or tool Like Torque, Moab or PBS. While, a typical HPC resource has several different users with various types of workloads, Hadoop workload is similar in nature. Each job that runs on HPC system can get different node configurations, can be placed in various toplogy configurations or can get different node types depends upon the type of hardware configuration, available queues and scheduling policies. The changes in the standard Hadoop cluster are very rare in terms of node configuration or node placements in the topology. We have integrated Yarn [37] with the Torque scheduler that is currently available on the system.

4.3 Challenges On Blue Waters

- **Scheduler:**
  User jobs on supercomputing systems are typically managed by a job management system and a resource manager. The Blue Waters system uses TORQUE Resource Manager integrated with the Moab Workload Manager to schedule and manage user jobs. Apache Hadoop stack comes with its own job scheduler, YARN (Yet Another Resource Negotiator). YARN expects to monitor and manage the nodes of a Hadoop cluster. The version of YARN we used in this paper does not integrate with existing workload and resource managers. This is an inherent conflict in how YARN and Blue Waters managers operate. MyHadoop works around this conflict as follows:
  - A regular job is submitted to the existing job scheduler
  - The list of nodes provided by the job scheduler are then used to create a set of configuration files
  - Using these files, a new (temporary) instance of Hadoop cluster is booted up
  - Hadoop jobs are not submitted to this instance of the Hadoop cluster

  In using this technique, other challenges were also encountered. Timeout values for various components had to be tuned to prevent the boot up process from failing. Another major issue was after a Hadoop job completes and tears down the Hadoop cluster, the BW scheduler failed to detect end of the job. This resulted in the BW scheduler waiting for wall clock timeout instead of terminating after job completes. The tear down process was modified so that BW scheduler could detect and release nodes for other jobs.

- **Scaling to larger node counts:**
  The Hadoop boot up script uses secure shell (ssh) to start Hadoop processes on each node. This is done in serial manner. For small node counts, this completed in a reasonable time. As we scaled to larger node counts (50+), the time to configure Hadoop cluster grew at an unacceptable rate. It is observed that he time taken for boot up is doubled when number of nodes are doubled. The time taken for 25, 50 and 100 nodes are shown in table 1.
  Serial tools exist that implement parallel remote shell. For our purposes, we have used *pdsh* [40] mainly because it is already configured to run on BW. Using pdsh instead of ssh, we noticed a significant improvement in boot up time.

Table 1: Timings of boot up using ssh on 25, 50 and 100 Nodes

Noof Nodes	Time
25	179.497
50	355.244
100	700.06

Table 2: Timings of BBP operations on 19 Nodes

Noof Maps	Computing Size	Time
100	0.5 x 10^6	782.53
100	1.0 x 10^7	3224.59
200	1.0 x 10^7	1806.07

5. Results

In this section we illustrate initial results for a spectrum of benchmarks that might give a broad picture of possibility of using HPC resouce such as Blue Waters for large-scale data analytics. We will perform extensive studies of other benchmarks and applications and look into scalabilty of these benchmarks in terms different Cluster Size as well as Data Sizes in detail in the future. We will also evaluate various possible optimization parameters on the system. We will not be running MapReduce / Hadoop workload on the entire system, instead we are planning on using upto 5% of the XE6 nodes which are more than 1000 nodes. If the results are encoraging we will perform scalability studies upto 2000 nodes for the purpose of this paper.

5.1 Performance Evaluation

In this section, we show the results of different standard Hadoop Benchmarks on the Blue Waters system. We also made sure that the experimental setup almost inline with the setup used in the work done at [30] so that we can have fare comparison of the obtained results. We also present the results that illustrate the impact of different data sizes and node sizes and scalability studies with respect to Data size and Cluster size.

5.1.1 PI Calculation

BBP is a map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of *PI*. This program is available with the Hadoop distribution. BBP is a purely computational application that employs a Bailey-Borwein-Plouffe method to estimate the value of *PI*. It is very nearly "embarrassingly parallel": the map tasks are all independent and the single reduce task gathers very little data from the map tasks. There is little network traffic or storage I/O. The results of BBP on 5 nodes is shown in figure 2. It shows that the time taken is decreasing when number of map tasks are increased. BBP is computing 0.5 x 10^6 digits. The time taken on our system is 25% less than the time taken in [30] paper. The results obtained for 19 nodes are shown in table 2. It shows that there is 56% improvement in time taken when number of Maps are doubled for the computation of 1.0 x 10^7 digits.

We also have performance numbers on more numbers of nodes that are encoraging. However we have not presented it here as we have not completed all the runs at this time.

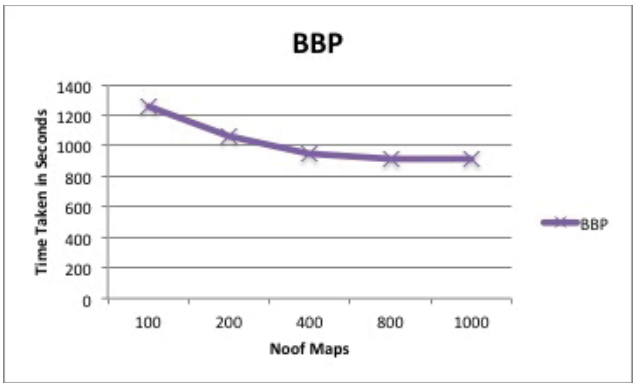

Fig. 2: Performance of BBP on 5 Nodes

5.1.2 Word Count

We have used two datasets for Word Count. One is Wikipedia [29] and other is Freebase [28]. The oroginal size of Wikipedia dataset was small, so we have duppli-cated the dataset few number of times to make the larger dataset of size 105GB. Freebase is an opensource datase released by Google. The size of this dataset is 361 GB. This dataset is a knowledge graph database for structuring human knowledge, which is used to support the collaborative web based data oriented applications. We have used 5 nodes for the performance of Word Count opeartion so that we can compare the results obtained in [30] with our results. The total time taken for wikipedia database is 2719 seconds and 4312 for Freebase database which is little more compare to [30] results. We will investigate further the reason behind this operation.

5.1.3 Grep

We have used Wikipedia [29] and is Freebase [28] datasets for the Text Search operation with datasizes 105 GB and 361 GB respectively. We have used 5 nodes for the performance of Text Search opeartion so that we can compare the results obtained in [30] with our results. The total time taken for Text Search Operation for Wikipedia dataset is 1019 seconds while for Freebase is 2884 seconds. There is more than 50% of reduction in the execution time on the JYC system as compare to results in [30]. We are confident that we will observe similar performance on Blue Waters too. On 20 nodes the time taken is 874 seconds and 300 seconds for Fressbase and Wikipedia dataset respectively. The performance is order of 3.5 magnitute improved with respect to the results obtained on 5 nodes.

5.1.4 TestDFSIO

DFSIO test handles large number of tasks performing read or write operations in parallel. In this test we have used 25, 50, 100, 200, 400 and 500 XE nodes. The total numbers of file written and read were 625, 1250, 2500, 5000, 10000 and

Table 3: Timings of Write and Read operations of DFSIO on 25, 50, 100, 200, 400 and 500 Nodes

Number Of Nodes	Write Oper	Read Oper
25	296.612	334.23
50	372.98	368.72
100	371.3	435.15
200	373.7	487.15
400	374.23	501.1
500	375.92	511.76

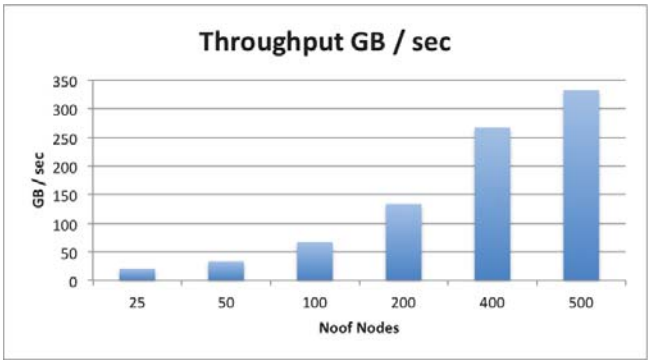

Fig. 3: Performance of DFSIO on on 25, 50, 100, 200, 400 and 500 Nodes

12500 respectively. The time taken for both Write and Read opeartions in mentioned in table 3. For 25 nodes we have observed 21.07 GB/Sec, for 50 nodes 33.51 GB/Sec, for 100 nodes 67.33 GB/Sec, for 200 nodes 133.79 GB/Sec, for 400 nodes 267.21 GB/Sec and for 500 nodes 332.51 GB/Sec. The figure shows that the throughput increases linearly as number of nodes are increased. The average IO rate is 187 MB/Sec for all the five configurations for write operations and varies between 159 MB/Sec to 175 MB/Sec for read opeartions. The throughput obtained is shown in figure 3. The default dfs.blocksize is 128 MB, therefore we have set the Lustre strip size to 128 MB and Lustre stripe count to 160, which is maximum number of OSTs to be used for the *scratch* file system.

6. Related Work

Benchmarking is a de-facto process to measure performance of any given system using a specific operation or set of programs to compare the achieved results with standard measures or other similar systems. Benchmarks are used not only to test but also to measure and to predict the sustained performance of computer system. Benchmarks are also used to reveal their architectural weakness and strong points. Benchmark data can provide valuable insight into the likely behavior of a given system; it may also be used to predict the performance of a new design. Benchmark data on the other hand reflect more specifically how appropriate the given design is for particular set of programs. Benchmarks can be classified according to application classes, such as scientific computing, commercial applications, distributed systems, network services, multimedia applications, and signal processing, etc. It is an important factor for evaluating distributed systems, and extensive work has been conducted in this area. There are various scientific and industry standard performance benchmarking programs available. Some of them are domain specific, some are associated with computer hardware or software systems.

One of the most popular benchmark suite is TPC benchmarks. The Transaction Processing Performance Council (TPC), a non-profit organizatio that defines transaction processing and database benchmarks, and distributes vendor-neutral performance data to the industry. They have several domain specific benchmarks such as TPC-C [5] and TPC-E [6] , an on-line transaction processing benchmark to evaluate online transaction processing (OLTP) performance on various hardware and software configurations, TPC-DS [8] and TPC-H [7], evaluates decision support systems, while TPC-App [9] is an application server and web services benchmark.

These benchmarks are useful in analysing performance of distributed systems, however they are not suitable to evaluate MapReduce framework. The scheduling policies [10], data replication and partitioning policies [11], [12] involve functionalities of microbenchmarks such as grep, word count and sort which are available with standard Hadoop distribution as described in [1].

There are few papers [13] and [14] depicting performance of MapReduce on parallel database systems. In [15], the authors compare MapReduce with parallel database system while in [16] authors study how the job configuration parameters affect the performance of Hadoop. In [17] , authors focus on architectural design issues and possible solutions to improve the overall performance of Hadoop.

In [18], authors discuss about the framework which is strongly based on myHadoop [19] approach to run Hadoop workload on HPC machines and initial results on 33 nodes of Cray XE6 / XK7 system. However none of them have performed detailed studies on the system as massive as Blue Waters, a Cray XE6/XK7 system consisting of more than 22,640 XE6 compute nodes (each containing two AMD Interlagos processors) augmented by more than 4224 XK7 compute nodes (each containing one AMD Interlagos processor and one NVIDIA GK110 "Kepler" accelerator) in a single Gemini interconnection fabric. While the results were obtained in [18] are using in memory for the Cray system, we will be using shared files system, Lustre.

In [23] authors discuss on optimizing nonblocking MPI [26] collective operations to optimize MapReduce and in [24] authors talk about a collective communication library, Harp that can be used to support various applications from HPC to cloud systems. In [25] authors have developed a high performance MapReduce system for the MPI environment that can be used to develop scientific applications in the

molecular dynamics fileld. In this paper, we are not focusing on performance of HPC application that use MPI extensively. However, we will look onto it in the future.

BlueWaters is one of the most powerful supercomputers in the world that provides a HPC platform for scientists and engineers across the country to solve wide range of challenging problems. We want to evaluate how such massive HPC resource can help solving large-scale data analytics, data-mining problems using MapReduce / Hadoop framework.

7. Conclusion and Future Work

There is no official support for Hadoop or related stack on the Blue Waters system. The first and most important challenge in achieving the project goal is to build a working Hadoop stack on the system. We have built a working stack on the Blue Waters system but it is not available to users. The focus of the paper is to obtain benchmarking results and not to provide a stable Hadoop stack on the system. Using the latest version may not be feasible due to the software ecosystem limitations. Therefore, the version works best within the system limitations will be used.

Hadoop works with a share-nothing architecture, where as systems such as BW are share everything designs. Using a shared file system like Lustre may pose challenges. Some workarounds are being investigated but their feasibility on Blue Waters system is unknown at this time. We have integrated YARN with the resource scheduler, MOAB that is available on the system.

Currently, we have only initial results with the existing opensource Apache Hadoop stack [2]. If this does not produce comparable results, we will consider Ohio State University's IB-enabled Hadoop stack [31]. Blue Waters can expose VERBS interfaces over Gemini network using IBoGNI. However, the stability, and compliance of IBoGNI is not well known. If it is stable, this stack will be preferred.

In this paper we have presented initial results of few benchmarks such as PI, Grep, TestDFSIO, NNBench etc. We will be considering few other standard benchmarks such as terasort, contrail bio workload etc. to perform detailed evaluation and analysis of MapReduce framework on Blue Waters. We may consider Intel's HiBench and PUMA benchmark suite and benchmarks if time permits. We will also evaluate MRBS benchmark suite provided the tarball is made available by the developers.

We will perform extensive studies of other benchamrks and applications and look into scalabilty of these benchmarks in terms different Cluster Size as well as Data Sizes in detail in the future. We will also evaluate various possible optimization parameters on the system.

The initial results on the MapReduce / Haddop framework are encoraging and we are confident that the massive traditional High Perfromance Computing resource can be useful in tackling the big-data research challenges and in solving large-scale data analytics, data-mining problems.

8. Acknowledgments

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. We would also like to thank to the anonymous reviewers and colleagues for their detailed and thoughtful comments and suggestions.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters", in Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, ser. OSDI 2004. Berkeley, CA, USA

[2] [Online]. Apache Hadoop Available: https://hadoop.apache.org/

[3] Blue Waters. Sustained Petascale Computing, NCSA, University of Illinois; 2014

[4] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, Bo Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis", ICDEW, 2010, 2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW) 2010, pp. 41-51

[5] [Online]. TPC-C. Available: http://www.tpc.org/tpcc/default.asp

[6] [Online]. TPC-E. Available: http://www.tpc.org/tpce/default.asp

[7] [Online]. TPC-H. Available: http://www.tpc.org/tpch/default.asp

[8] [Online]. TPC-DS. Available: http://www.tpc.org/tpcds/default.asp

[9] [Online]. TPC-App. Available: http://www.tpc.org/tpc_app/default.asp

[10] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, âĂIJImproving MapReduce Performance in Heterogeneous Environments,âĂİ in 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI2008), 2008.

[11] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, âĂIJScarlett: Coping with Skewed Content Popularity in MapReduce Clusters,âĂİ in EuroSys 2011 Conference, Salzburg, Austria, April 2011.

[12] M. Eltabakh, Y. Tian, F. Ozcan, R. Gemulla, A. Krettek, and J. McPherson, âĂIJCoHadoop: Flexible Data Placement and Its Exploitation in Hadoop,âĂİ in 37th International Conference on Very Large Data Bases (VLDB 2011), Seattle, Washington, September 2011

[13] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In SIGMOD , pages 165(78. ACM, 2009)

[14] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? Communications of the ACM , 53(1):64(71, 2010)

[15] S. Babu. Towards automatic optimization of mapreduce programs. In SoCC , pages 137-142. ACM, 2010

[16] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. Commun. ACM , 53(1):72-77, 2010.

[17] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. 2010. The performance of MapReduce: an in-depth study. Proc. VLDB Endow. 3, 1-2 (September 2010), 472-483.

[18] Scott Michael, Abhinav Thota and Robert Henschel, HPCHadoop: A framework to run Hadoop on Cray X-series supercomputers, *Cray User Group Meeting 2014*

[19] S. Krishnan, M. Tatineni, and C. Baru, âĂIJmyHadoop - Hadoop-on-Demand on Traditional HPC Resources. Accessed 04-28-2014.âĂİ http://www.sdsc.edu/ãllans/MyHadoop.pdf

[20] [Online]. Available: http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench#testdfsio

[21] Sangroya, Amit and Serrano, DamiÃąn and Bouchenak, Sara, MRBS: A Comprehensive MapReduce Benchmark Suite. Technical Report (RR-LIG-024), LIG Laboratory, University of Grenoble, Feb 2012.

[22] [Online]. Available: http://hadoop.apache.org/docs/current/api/-org/apache/hadoop/examples/pi/package-summary.
[23] Torsten Hoefler, Andrew Lumsdaine, and Jack Dongarra. 2009. Towards Efficient MapReduce Using MPI. In Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Matti Ropo, Jan Westerholm, and Jack Dongarra (Eds.). Springer-Verlag, Berlin, Heidelberg, 240-249.
[24] Zhang, Bingjing, Yang Ruan, and Judy Qiu. Harp: Collective Communication on Hadoop. Technical Report Indiana University, 2014.
[25] Matsuda, Motohiko, Naoya Maruyama, and Shin'ichiro Takizawa. "K MapReduce: A scalable tool for data-processing and search/ensemble applications on large-scale supercomputers." Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE, 2013.
[26] Message Passing Interface Forum: MPI: A Message Passing Interface Standard.
[27] [Online]. Available: http://hortonworks.com/blog/how-to-plan-and-configure-yarn-in-hdp-2-0/
[28] [Online]. Available: https://developers.google.com/freebase/data
[29] [Online]. Available: http://dumps.wikimedia.org/enwiki/latest/
[30] Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali R. Butt, and Nicholas Fuller. 2014. MRONLINE: MapReduce online performance tuning. In Proceedings of the 23rd international symposium on High-performance parallel and distributed computing (HPDC 14). ACM, New York, NY, USA, 165-176
[31] [Online]. Available: http://hibd.cse.ohio-state.edu/
[32] Frank Schmuck and Roger Haskin. 2002. GPFS: A Shared-Disk File System for Large Computing Clusters. In Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02). USENIX Association, Berkeley, CA, USA, , Article 19
[33] [Online]. Available: http://en.wikipedia.org/wiki/IBM_General_Parallel_File_System
[34] [Online]. Available: http://lustre.org/
[35] HPC Scheduling and Job Prioritization. [Online]. Available: https://www.adaptivecomputing.com/wp-content/uploads/2014/06/HPC_Scheduling_White.pdf
[36] TORQUE Resource Manager [Online]. Available: http://www.adaptivecomputing.com/products/open-source/torque/
[37] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: yet another resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (SOCC '13). ACM, New York, NY, USA, , Article 5 , 16 pages.
[38] C. Clos, "A Study of Non-blocking Switching Networks", Bell Systems Technical Journal, vol. 32, 1953.
[39] David H. Bailey, Peter B. Borwein and Simon Plouffe, "On the Rapid Computation of Various Polylogarithmic Constants", Mathematics of Computation, vol. 66, no. 218 (Apr 1997), pg. 903 - 913.
[40] [Online]. pdsh. Available: http://code.google.com/p/pdsh/