# HPCHadoop: A framework to run Hadoop on Cray X-series supercomputers

Scott Michael, Abhinav Thota, and Robert Henschel
Pervasive Technology Institute
Indiana University
Bloomington, IN, USA
Email: scamicha@iu.edu

***Abstract*—In recent years a variety of Big Data challenges have arisen in both research and industry. These new challenges have been met by the development of software frameworks, such as the MapReduce framework. Although there are many implementations of the MapReduce framework, the most widely used open source implementation is Apache Hadoop. Even though there are many Big Data research challenges being tackled with HPC resources that could benefit from a MapReduce approach, Hadoop or other MapReduce frameworks are seldom deployed in HPC centers, and never, to our knowledge, on a Cray X-series supercomputer. In many cases, the only large compute resources that researchers at universities and government labs have access to are HPC machines. Moreover, there are many cases where researchers would like to run existing Hadoop codes on HPC machines without having to convert their codes to another parallel framework. This creates the need for a Hadoop solution on HPC machines.**

**In this paper we present a framework, called HPCHadoop, to enable researchers to run Hadoop workloads on HPC machines. We detail the design of the framework and present some preliminary benchmark data. HPCHadoop is specifically targeted to enable Hadoop workloads on the Cray X-series supercomputers (XE, XK, and XC), but can be used on any supercomputing platform. We present preliminary results from the Intel HiBench Hadoop benchmark suite for varying levels of parallelism and for two hardware configurations Big Red II, a Cray XE/XK system, and Quarry, a traditional gigabit connected cluster. We compare the performance of the framework for a variety of storage backends, including node local memory (RAM) storage, node local hard drive storage, and a shared Lustre filesystem used as either a direct file store or with HDFS layered over it.**

***Index Terms*—Data Analysis, Data Storage Systems, Data Mining**

## I. Introduction

In the past several years, we have seen the rise of an array of techniques and methodologies to extract valuable information from a class of data sets identified as "Big Data". One extremely prevalent class of techniques is MapReduce [1]. Arguably the most widely used distribution of the MapReduce paradigm is the Apache Hadoop MapReduce framework [2]. Although the Hadoop framework has seen wide adoption and usage throughout industry and academia [3], it has yet to take hold as a typical use case at an HPC center. For example, to our knowledge, there are no Hadoop distributions that support running the Hadoop framework on a Cray X-series machine, and only a few tools to run the Hadoop framework on a more general HPC platform [4], whose current usage and support are questionable.

In general, there are two major obstacles for a user to overcome when deploying and using the Hadoop framework on an HPC resource. First, a traditional Hadoop cluster is typically entirely devoted to Hadoop workloads, and the resource management and scheduling is handled by the Hadoop framework. In contrast, a typical HPC system serves a very broad user base, and executes a variety of workloads at any given time. Resource scheduling and management is usually handled by dedicated software such as PBS/Moab or Slurm. This means that for each job a user runs, they may get different nodes in the cluster, both in terms of network location and, depending on the queues and available node types in the cluster, individual node configurations. This is very different from a standard Hadoop cluster configuration, which is static or changes only very rarely both in terms of the cluster networking and configuration of individual nodes. Deploying the Hadoop framework within an environment managed by such dedicated software is not straightforward or transparent to end users. The Cray X-series adds the additional wrinkle that the Hadoop framework requires services that are provided by Cluster Compatibility Mode (CCM) mode, so the system must support CCM mode and the Hadoop framework should be properly launched via the `ccmrun` command.

Secondly, Hadoop employs a "shared nothing" architecture. Specifically, the Hadoop Distributed File System (HDFS) [5] assumes each node in the cluster has its own local data storage, and there is no shared data storage system. The HDFS system then provides data distribution and replication throughout the node local storage in the cluster. This is in stark contrast to many HPC centers, the majority of which have a high performance distributed storage systems, such as Lustre or GPFS [6], that is shared among all the nodes in a cluster, or even among several clusters at a center. While some machines at HPC centers may have node local storage in addition to a global high performance filesystem, many HPC centers have clusters that do not have dedicated node local data storage. The Cray X-series, in particular, has no node local storage.

In this paper we present the design of the HPCHadoop framework. HPCHadoop is a framework developed at the Indiana University Pervasive Technology Institute, designed to simultaneously address the aforementioned obstacles and

enable users to run Hadoop workloads on HPC systems. It is specifically targeted to enable Hadoop workloads on the Cray X-series supercomputers (XE, XK, and XC), but can be used on any supercomputing platform. We also present the results of the Intel HiBench Hadoop benchmark suite for a variety of Hadoop workload sizes and levels of parallelism for several hardware configurations including Big Red II, a Cray XE/XK system, and Quarry, a traditional gigabit connected cluster. We compare the performance of the framework for a variety of storage backends, including node local memory (RAM) storage, node local hard drive storage, and a shared Lustre filesystem over both gigabit and Gemini interconnects. The balance of the paper is presented as follows: section II explains the key drivers in the design of the HPCHadoop framework and the steps to implement it, section III outlines the methodology used to set Hadoop tunables and perform benchmark runs. In section IV we present the benchmark results and follow this with some discussion of the results in section V. Finally, we present the subject of future development and studies in section VI and discuss a few final conclusions in VII.

## II. HPCHadoop: Design and Implementation

The design goals for HPCHadoop are simply to address the issues raised in section I around running the Hadoop framework in a batch scheduled HPC environment. The current incarnation of HPCHadoop is available on Github[1] and is principally intended as a proof of concept, though it does address the two major hurdles outlined in the Introduction. Beyond addressing the issues of integrating with a batch scheduling system and using nodes without node local storage, the system should require minimal configuration to get the Hadoop framework up and running on a Cray system. Hadoop has a nearly innumerable number of tunables and, for the most part, HPCHadoop does not expose these tunables to the end user. This is not to say that a user can not tune their Hadoop installation, merely that there is no facility provided within HPCHadoop to perform this configuration. In the end, the user is required to provide a Hadoop installation and modify three files (one of which is the job submission script) to configure and run a Hadoop job with HPCHadoop.

Although there are other frameworks available which can address some of the issues outlined, to our knowledge there was not previously a solution that addressed all the issues and allowed for deployment of the Hadoop framework on a Cray X-series machine. We have integrated several of the best features from several of these frameworks. For example, the myHadoop framework [4] was designed to allow for deployment of the Hadoop framework on HPC machines, and HPCHadoop is strongly based on the myHadoop approach. However, the fact that Hadoop needs to be launched in the CCM environment on the Cray X-series presented challenges for several of the implementation details of myHadoop. In general, experience has shown that the most effective way of launching CCM jobs is to `ccmrun` a script, which then may do setup, or launch other scripts. This design, along with the basic approach of myHadoop was ultimately how HPCHadoop was implemented.

In detail, HPCHadoop is implemented as a series of scripts that are invoked by `ccmrun` and interact with the batch scheduler to configure and launch the Hadoop cluster. Currently HPCHadoop supports the PBS scheduler, although adding additional schedulers, such as Slurm, is a relatively straightforward proposition. The end user has minimal configuration requirements, needing only to edit three files. The first of which defines several environment variables in the HPCHadoop installation including the location of the Hadoop installation, the data location, the type of filesystem used, and so forth; the second provides the Hadoop commands to be run; and the third being a PBS job submission script file.The system supports creating an HDFS instance on either node local storage (the `/tmp` filesystem in local memory in the case of the Cray X-series) or on a globally shared filesystem such as Lustre. Prototype functionality to directly use a global filesystem, without the HDFS layer is also included, but has not been fully validated, and so benchmark results are not presented in this paper.

Once the end user has set the appropriate variables and included the necessary files, they can submit their job to the batch scheduling system. Following the start of the job and instantiation of the CCM environment on the job nodes, the HPCHadoop framework retrieves the node list and populates the slave and master lists in the Hadoop installation. The Hadoop configuration files are also populated with the user defined variables. The HDFS file system is then instantiated and the Hadoop framework is initiated. Following the successful startup of the Hadoop framework, the user's Hadoop workload is executed and progress is logged to the user specified log directories. Upon completion of the Hadoop workload, the HPCHadoop framework preserves the log data, cleans up the Hadoop framework on the job nodes and terminates the job.

## III. Methodology

In this section, we describe our approach to testing the HPCHadoop framework and comparing the performance of Hadoop and HPCHadoop on different hardware architectures. After careful consideration, we chose the Intel Hadoop Benchmark Suite (HiBench) version 2.2 [7] to be the workload for our tests. HiBench was paired with HPCHadoop running Apache Hadoop version 1.2.1. HiBench contains 9 typical Hadoop workloads, including Bayes, DFSIOE, Kmeans, Nutchindex, Pagerank, Terasort and Wordcount.

There are many different parameters that can be tweaked in both the Hadoop framework and within the Hibench Suite to compare performance and behavior, including data size, number of maps and reduces, running in memory versus on native Lustre and on Lustre without HDFS. In this work, we decided that it was out of scope to go beyond minimal tuning of the many Hadoop and benchmark specific tunables. Our aim with the benchmark numbers is to provide some comparison between a Cray X-series machine and a more

[1] https://github.com/scamicha/HPCHadoop

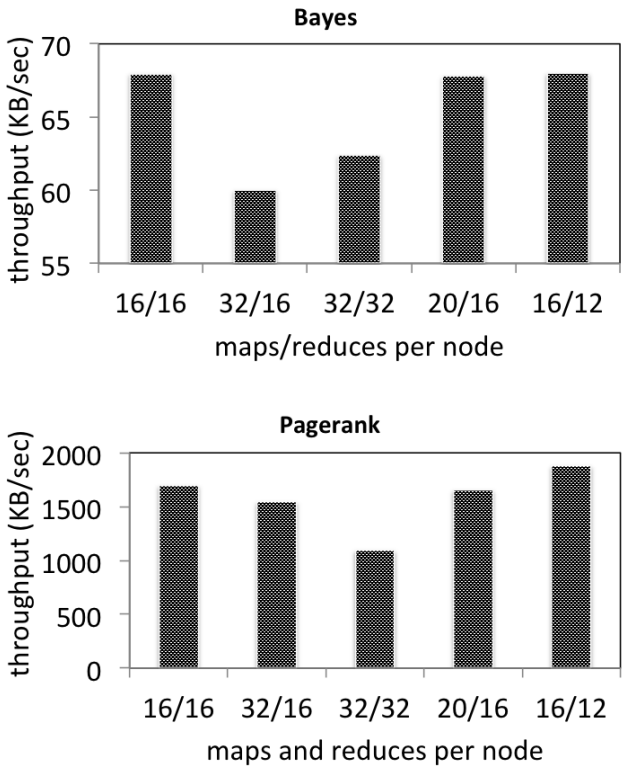


Fig. 1: Both plots show the single node throughput in kilobytes per second for two sample benchmarks, Bayes and Pagerank, for a variety of mapper and reducer combinations. The benchmarks were run on the XK7 nodes (16 cores/node) of Big Red II.

traditional style Hadoop like cluster. However, we determined that at minimum we needed to find a good ratio of maps and reduces for the best performance on the given hardware configurations. Following this determination, we determined fixed input data sizes to use for each of the benchmarks, which was largely based on the minimum input data size that would reasonably demonstrate scaling. We then ran a series of benchmarks to study the scalability of the HPCHadoop and Hadoop frameworks on both the Cray machine and our standard cluster. These benchmarks were run on both local storage (in memory in the case of the Cray) and on a globally shared Lustre filesystem with HDFS.

### A. Hardware

The following is a description of the hardware used in all of the benchmark tests presented throughout the balance of this paper. Big Red II is a Cray XE6/XK7 hybrid supercomputer at Indiana University, with a single AMD Interlagos 16 core CPU, 32 GB of memory, and a single NVIDIA K20 GPU per XK7 node and two AMD Abu Dhabi 16 core CPUs and 64 GB of memory per XE6 node. The nodes are interconnected via the Cray Gemini network, which is connected to a shared Lustre file system called the Data Capacitor II (DC2) via 22 LNET routers. Quarry is a standard Linux cluster, with two Intel based 8 core CPUs and 16GB of memory per node. Each Quarry node also contains a local disk with a usable capacity of 100GB. Quarry nodes are connected via gigabit ethernet and are connected to DC2 via four LNET routers.

### B. Number of maps and reduces per node

To conduct a reasonable comparison of benchmark runs, we performed a minimal amount of tuning of the Hadoop framework, namely tuning the number of active mappers and reducers for a given task and number of cores in a worker node. We attempted runs of the HiBench framework with a variety of ratios of mappers to reducers. We tested the following ratios on the XK7 nodes of Big Red II: 16/16, 32/16, 32/32, 20/16 and 16/12. An HDFS instance instantiated on the DC2 filesystem was used for all of these runs. For most of the benchmarks, a 16/12 maps to reduces ratio gives the best or close to best runtime. The closest competitor to 16/12 is 16/16. A sampling of the results from the runs to find the best ratio of maps and reduces are shown in figure 1, here we only show the results for two of the benchmarks involved, Bayes and Wordcount. However, the results from the other benchmarks are similar. Based on this finding, for all subsequent benchmark runs discussed in this paper, we set the ratio of mappers to reducers to 4:3, with as many maps as there are cores per node.

### C. Input data size

Following the determination of the number of mappers and reducers per node, the other benchmark parameter we tuned was input data size, we determined an appropriate size that was not too small for runs with larger node counts and at the same time not too large for runs with smaller node counts. Again, we settled on a value that worked for most of the benchmarks involved. We experimented with a variety of data sizes, such as 2X, 2.5X, 5X and 10X times the default HiBench input data size. These input data sizes were not really an issue for runs on Lustre, other than to inflate the benchmark run times, but large data sizes were an issue when running in node local memory on the Cray. We finally chose 2.5X the original data size for each of the benchmarks as the standard for our performance comparison runs. This means that the smallest node count which we could run almost all of the benchmarks in memory was 5 nodes. We were not able to run node local memory benchmarks with 3 nodes. The input data sizes ranged anywhere from 500 MB to 250GB, depending on the benchmark. Also, for some of the benchmarks, the input data size was defined on a per node basis. Most of the benchmarks that are part of HiBench did not work out of the box. Many of them needed minor adjustments, including setting some missing environment variables and removing a few unnecessary statements and unbound variables.

To understand the performance and scaling characteristics of the various benchmarks, we designed a series of tests, on Big Red II and Quarry. We did scaling tests for each of the benchmarks on both the machines, going from 5 nodes to 33

nodes. These node counts were determined based on the fact that HPCHadoop by default configures the Hadoop cluster to have one node dedicated to the job tracker with the rest of the nodes as task trackers. These node counts allowed us to increase the number of task tracker nodes in powers of two.We repeated these tests both in memory and on Lustre for the Cray system and on the node local disk and on Lustre for the gigabit connected cluster. We discuss the results in the next section.

## IV. Results

In this section, we give a sampling of results designed to illustrate the extremes of the overall results. By bracketing the behavior of the spectrum of benchmark results we hope to give a comprehensive picture of what is possible with Hadoop in general and HPCHadoop specifically. It is worth bearing in mind that minimal amounts of Hadoop specific and per benchmark specific optimizations have been performed to obtain these results, so they are by no means the maximum that could be expected from Hadoop and HPCHadoop. We examine the scalability of the framework, looking at both the benchmarks exhibiting the greatest and least measures of scalability on both Big Red II and Quarry. This is followed by a direct comparison of the runs on Big Red 2 and Quarry, again focusing on the benchmarks that show both the largest variation in performance and the least variation.

We have adopted this approach because we did not observe any particularly strong trends that were consistent across all of the benchmarks, the benchmarks vary widely in their underlying workloads, measured throughput, and scalability. With further and more detailed investigation, and per benchmark fine tuning it may be possible to measure some more general trends, but many of the benchmarks in the HiBench suite are actually designed to measure fundamentally different system characteristics. By bracketing the overall behavior, we are attempting to give the reader a broad sense of the spectrum of possibilities for all the benchmarks that fall somewhere in between the extremes. Although we did some initial tests on the XK7 nodes of Big Red II, all the results described in this section are from runs on the XE6 nodes of Big Red II.

### A. Scaling on Cray X-Series

Figure 2 shows the scaling behavior of two benchmarks on Big Red II, for both the local memory benchmarks and on the DC2 filesystem. The top plot shows the scalability of the Nutchindexing benchmark, presenting the aggregate throughput of the benchmark as a function of the number of nodes used. One can observe that in going from 5 to 33 nodes, the throughput stays roughly constant, with only a slight increase in total throughput as the number of nodes is increased. The behavior of the Nutchindexing benchmark is an extreme example of a benchmark that did not scale well as the number of nodes was increased.

The bottom graph of figure 2 shows the same sort of information for the Wordcount benchmark. In stark contrast to the Nutchindexing benchmark, the Wordcount benchmark is much more scalable. The throughput goes up nearly five times from 225 MB/sec to 956 MB/sec in the case of the in memory run and goes up about three times from 211 to 602 MB/sec with DC2.

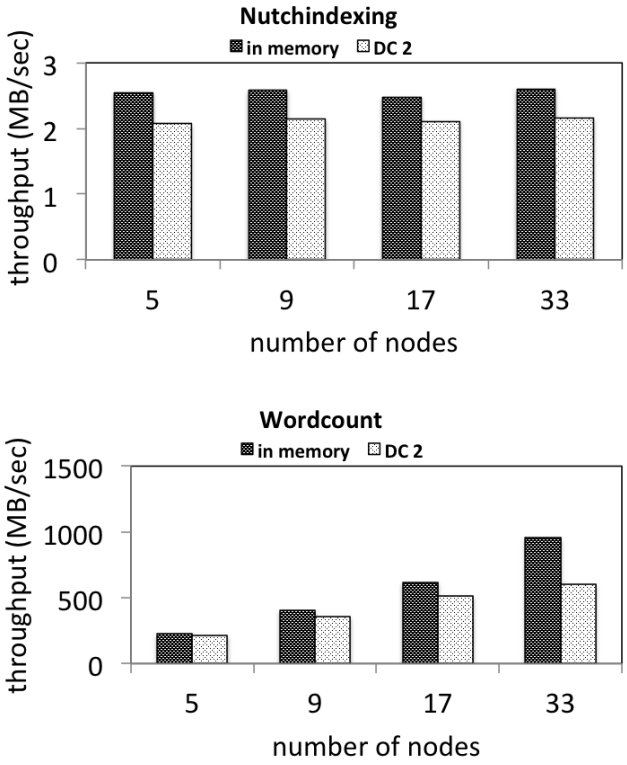


Fig. 2: **Big Red II:** The top plot shows the throughput of the Hadoop framework in megabytes per second for the Nutchindexing benchmark, the benchmark exhibiting the poorest scalability, on Big Red II for a variety of node counts on both node local memory storage and Lustre storage. The bottom plot shows the same values for the Wordcount benchmark, which is the benchmark that exhibits the best scaling on Big Red II.

It is worth noting that the input data size is specified on a per node basis for Wordcount, and therefore the aggregate input data size increases along with the number of nodes, such that the total input data size is 256 GB for the 33 node benchmarking run. On the other hand the Nutchindexing input data set is constant for increasing node count and is relatively small, only 1.5 GB, but, even so, should be enough to exhibit some measure of scaling at smaller node counts.

### B. Scaling on Conventional Hadoop Cluster

We carried out a similar set of experiments on the Quarry cluster, which is a machine with a hardware configuration that is closely matched to a conventional Hadoop cluster. For the sake of comparison, we selected the same benchmarks for discussion of the scaling behavior on Quarry. The key difference between the Big Red II runs and Quarry runs that utilized local storage used node local memory on Big

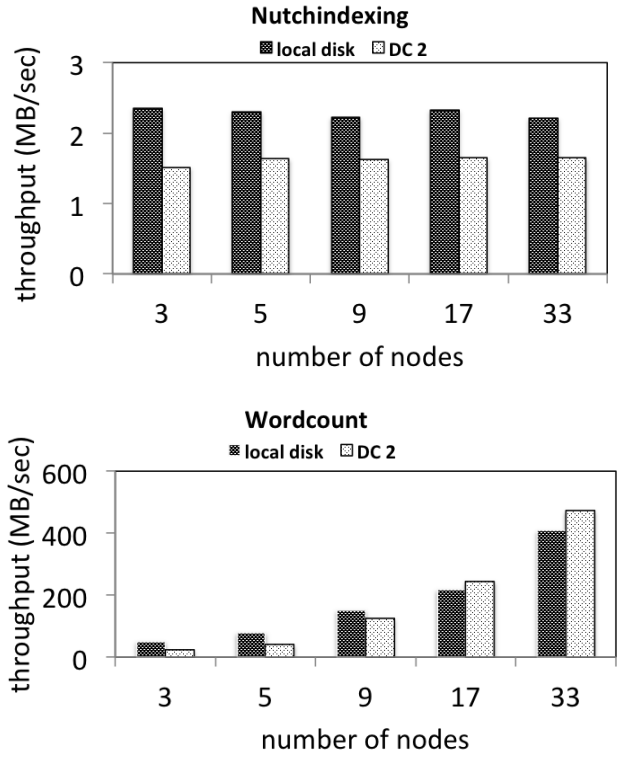


Fig. 3: **Quarry:**The top plot shows the throughput of the Hadoop framework in megabytes per second for the Nutchindexing benchmark, the benchmark exhibiting the poorest scalability, on Quarry for a variety of node counts on both local disk storage and Lustre storage. The bottom plot shows the same values for the Wordcount benchmark, which is the benchmark that exhibits the best scaling on Quarry.

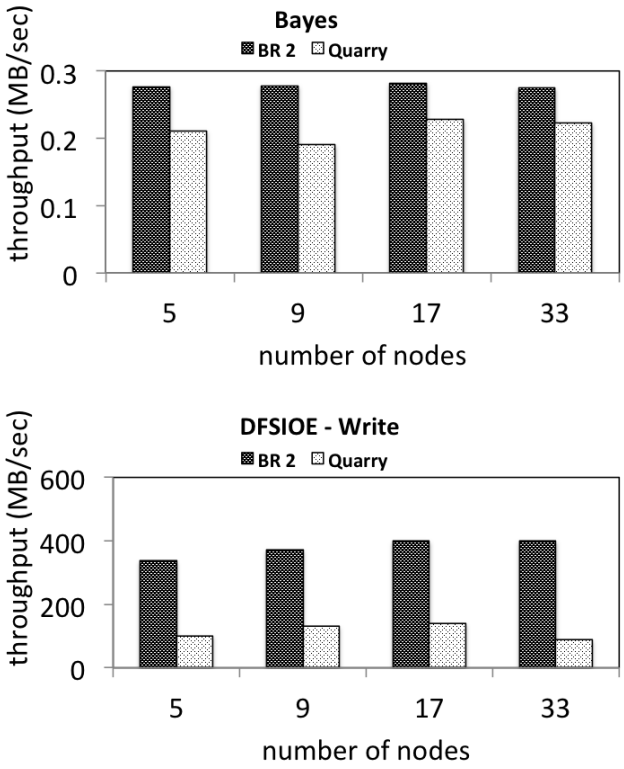


Fig. 4: **Local Storage:** The top chart shows the Bayes benchmark, the benchmark with the smallest difference in performance between Big Red II and Quarry for local storage. Conversely, the bottom chart shows the DFSIOE benchmark, the benchmark with the largest difference in performance. The Big Red II runs used node local memory and the Quarry runs utilized a local disk attached to the compute nodes.

Red II and local disks for the Quarry nodes. This was done because each Quarry node has only 16GB of memory, which is insufficient for a majority of the benchmarks. And, of course, Big Red II nodes have no local disks. The runs on DC2 were carried out in fashion similar to the DC2 runs performed on Big Red II.

Figure 3 shows the scaling behavior of Nutchindexing and Wordcount on Quarry when the number of nodes is increased from 3 to 33. We were able to fit and run the benchmarks on 3 Quarry nodes because of the larger disk space that is available on the local disks. As with Big Red II, Nutchindexing in the top panel is an extreme example of a benchmark that does not scale at all with an increase in the number of nodes. Wordcount is an example from the other end of the spectrum, and shows consistently good scaling from 3 to 33 nodes. It is worth noting that the aggregate Wordcount throughput for Quarry is a factor of two or more smaller than that of Big Red II. This is most likely due to the fact that Quarry has only 8 cores per node, compared to the 32 on Big Red II.

### *C. Comparison Between Cray X-series and Conventional Hadoop Cluster*

By examining the individual benchmarks which show both the greatest and least performance differences on Big Red II as compared to Quarry one can get a sense of the range of possibilities in performance when looking at a Cray X-series machine versus a conventional Hadoop cluster. The runs on Big Red II and Quarry use the same configuration and input data, one major difference being Big Red II has 32 cores per node and Quarry has 8 cores per node. Another detail to keep in mind is that, for local storage, we are comparing in memory runs on Big Red II with the runs that utilize the local disk on Quarry. We also compare the runs that use HDFS on DC2.

*1) Local Storage:* Figure 4 shows how the performance of Bayes and DFSIOE-Write on Big Red II compare with Quarry. The throughput of the Bayes benchmark on Big Red II and Quarry differ by less than 20%. This is the smallest performance differential measured for any of the benchmarks when using local storage. On the other end of the spectrum, one can observe in the bottom plot of figure 4 that the DFSIOE-Write runs on Big Red II are consistently more

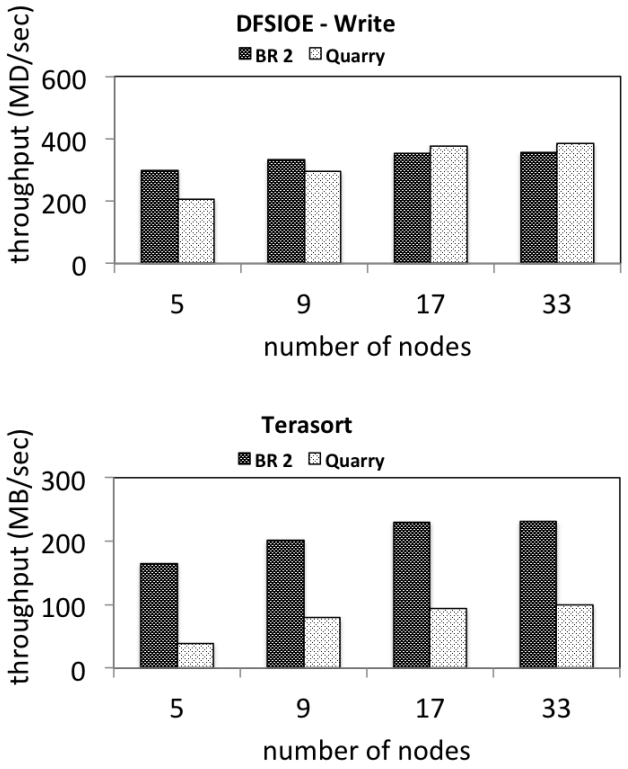


Fig. 5: **DC2**: The top chart shows the DFSIOE benchmark, the benchmark with the smallest difference in performance between Big Red II and Quarry for Lustre storage. Conversely, the bottom chart shows the Terasort benchmark, the benchmark with the largest difference in performance. Both the Big Red II runs and the Quarry runs used the DC2 filesystem.

than three times faster than the Quarry runs using node local storage. All the other benchmark performance differentials for local storage fall between DFSIOE-Write and Bayes in how they compare between Big Red II and Quarry.

*2) Global Lustre Filesystem:* Figure 5 shows two benchmarks, one that performs at roughly the same level on both Big Red II and Quarry and the other that varies by a large margin. These runs used HDFS instantiated on the DC2 Lustre filesystem. The graph on the top shows the DFSIOE-Write benchmark, whose performance does not differ very much between BR 2 and Quarry. The graph on the bottom shows the Terasort benchmark, in which Big Red II consistently outperforms Quarry. These two benchmarks represent the opposite ends of the spectrum and all the other benchmarks fall between these two in the comparison between Big Red II and Quarry.

## V. Discussion

In all, we performed 20 runs of the HiBench suite, for a total of 160 individual benchmark results. The results presented in section IV are intended to be instructive in that they bracket the behavior of the benchmarks in scaling and when comparing the Cray system to a traditional Hadoop cluster. Although the results are fairly wide ranging, one can draw several general conclusions from them regarding the typical performance one could expect from a Cray X-series machine.

It should be noted that our initial main objective with the HPCHadoop framework was to produce an easy to use, straightforward, proof of concept implementation of a framework that would allow users to make use of the Hadoop framework on Cray machines. By this measure, HPCHadoop has met our initial goals, in that it was able to successfully run many iterations of a standard benchmark suite on both the Cray platform and a standard Linux cluster. Of course, one may first ask the question whether it even makes sense to attempt to deploy Hadoop on an HPC cluster. In general, the architecture and design of most HPC systems is fundamentally at odds with the basic assumptions underlying the Hadoop framework. However, there are several use cases one can imagine where a researcher would want to use the Hadoop framework on an HPC resource.

The first such instance is when a researcher has access to already developed Hadoop code that they wish to apply to their own data sets. In this case there is a great deal of benefit to the researcher to not have to develop code from scratch to work in a different distributed algorithm paradigm. Additionally, if a researcher wishes to make use of other computational resources besides HPC centers, such as Amazon or other cloud offerings, it is of great benefit to have a code base that works equally well on both architectures. A second potential use case lies in the fact that the MapReduce paradigm is a very simple way to introduce researchers to a parallel distributed computing paradigm, particularly when their workflow is data parallel. A HPC enabled Hadoop framework offers the possibility of offering an extremely simplified parallel processing service, where the researcher is only required to provide a map function and a reduce function. To be sure, this limits the scope of the type of research problems that such a framework can address, but it also dramatically lowers the barrier for entry.

For certain classes of problems Hadoop on a Cray offers the potential of good scalability right out of the box, as is evidenced by the best scaling benchmarks in figure 2. Although some types of workloads may require some further fine tuning, these are most likely issues that can be addressed fairly easily. We also found that, in general, a Cray X-series system outperforms a standard Hadoop cluster. In some cases, as in figures 4 and 5, by factors of several. Clearly, with a framework such as HPCHadoop a Cray X-series machine offers an opportunity to address Big Data problems with the Hadoop framework.

## VI. Future Work

The development of HPCHadoop has proceed fairly rapidly and the overall intention of this paper was to act as a proof of concept for the framework. Although the framework has been shown to successfully run all of the benchmarks in the HiBench suite, there remain many areas for potential improvement or further development. For example, although

we have a working prototype which allows a user to run Hadoop natively on Lustre (without starting HDFS), this model is still unstable and has only been shown to work with a few of the benchmarks and certain input data sizes. This work is very similar to the work conducted by Intel in their Intel Distribution of Hadoop (IDH). The IDH distribution contains a native Lustre connector which purports to allow users to directly use a shared global Lustre filesystem for Hadoop workloads. We have begun conversations with Intel to evaluate their product and approach and compare it to our own. We are also in the process of evaluating other systems that allow users to use globally shared filesystems such as Mariane [8].

In addition, to the evaluation of native filesystem protocols, we will revisit the benchmark parameters of the HiBench suite to attempt to better optimize the individual benchmarks and gain deeper insight into the strengths of the Cray X-Series for Hadoop workloads, following this track we will also investigate some novel Hadoop frameworks, such as Twister [9], to determine their relative strengths on the Cray hardware.

Finally, a relatively straightforward addition is to include support for other schedulers beyond the PBS system that is currently supported. We will begin by including HPCHadoop support for the Slurm scheduler.

## VII. Conclusion

We developed a straightforward implementation of an easy to use and easy to adapt framework, which can be used to run Hadoop jobs, specifically on Cray X-series supercomputers and, more generally, on regular batch scheduled Linux supercomputers. We have presented the results of several scaling runs of the HiBench benchmark suite for both local storage and HDFS on a Lustre filesystem on a Cray and a regular Linux cluster.

We gathered a large set of data from all the benchmarks, and compared the Hadoop framework throughput for local storage and Lustre on both Big Red II and Quarry. As can be observed from the results, we saw a broad spectrum of performance results, but in most cases local storage performed slightly better than Lustre, which is certainly understandable when memory is being use as node local storage. However, even given the disparity in the raw performance numbers it is surprising that the performance on both local storage and Lustre are in the same ballpark.

The comparisons between Big Red II and Quarry are more predictable, and Big Red II handily outperforms Quarry more often than not, whether using local storage or Lustre. It is difficult for us to say why each of the benchmarks behave the way they did, further study of each of the individual benchmarks and some deeper expertise in Hadoop benchmarking is required. However, we have put the HPCHadoop framework through its paces by running a wide variety of benchmarks with it and confirming that it does indeed support a broad variety of Hadoop workloads.

## 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'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264

[2] Apache Foundation, "Apache Hadoop - Official Site. Accessed 04-28-2014." http://hadoop.apache.org.

[3] ——, "Powered By Apache Hadoop. Accessed 04-28-2014." http://http://wiki.apache.org/hadoop/PoweredBy.

[4] S. Krishnan, M. Tatineni, and C. Baru, "myHadoop - Hadoop-on-Demand on Traditional HPC Resources. Accessed 04-28-2014." http://www.sdsc.edu/~allans/MyHadoop.pdf.

[5] Apache Foundation, "HDFS Architecture Guide," http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html.

[6] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083323.1083349

[7] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, 2010, pp. 41–51.

[8] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan, "Mariane: Using {MApReduce} in {HPC} environments," *Future Generation Computer Systems*, vol. 36, no. 0, pp. 379 – 388, 2014, special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13002719

[9] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 810–818. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851593