myHadoop - Hadoop-on-Demand on Traditional HPC Resources

Sriram Krishnan
San Diego Supercomputer Center
9500 Gilman Dr MC0505
La Jolla, CA 92093-0505, USA
sriram@sdsc.edu

Mahidhar Tatineni
San Diego Supercomputer Center
9500 Gilman Dr MC0505
La Jolla, CA 92093-0505, USA
mahidhar@sdsc.edu

Chaitanya Baru
San Diego Supercomputer Center
9500 Gilman Dr MC0505
La Jolla, CA 92093-0505, USA
baru@sdsc.edu

ABSTRACT

Traditional High Performance Computing (HPC) resources, such as those available on the TeraGrid, support batch job submissions using Distributed Resource Management Systems (DRMS) like TORQUE or the Sun Grid Engine (SGE). For large-scale data intensive computing, programming paradigms such as MapReduce are becoming popular. A growing number of codes in scientific domains such as Bioinformatics and Geosciences are being written using open source MapReduce tools such as Apache Hadoop. It has proven to be a challenge for Hadoop to co-exist with existing HPC resource management systems, since both provide their own job submissions and management, and because each system is designed to have complete control over its resources. Furthermore, Hadoop uses a shared-nothing style architecture, whereas most HPC resources employ a shared-disk setup. In this paper, we describe *myHadoop*, a framework for configuring Hadoop *on-demand* on traditional HPC resources, using standard batch scheduling systems. With myHadoop, users can develop and run Hadoop codes on HPC resources, without requiring root-level privileges. Here, we describe the architecture of myHadoop, and evaluate its performance for a few sample, scientific use-case scenarios. myHadoop is open source, and available for download on SourceForge.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*batch processing systems, distributed systems*
D.2.8 [**Programming Techniques**]: Concurrent Programming—*distributed programming*

General Terms

Management, Performance, Design, Experimentation

Keywords

MapReduce, Hadoop, High Performance Computing, Resource Management, Clusters, Open Source, myHadoop

1. INTRODUCTION

Traditional High Performance Computing (HPC) resources, such as those available on the TeraGrid [15], support batch job submissions using Distributed Resource Management Systems (DRMS) such as TORQUE [9] (also known by its historical name Portable Batch System - PBS), or the Sun Grid Engine (SGE - [7]). These systems are put in place by system administrators on these resources to enable submission, tracking, and management of batched, non-interactive jobs, such that it maximizes the overall utilization of the system, and that it enables sharing of the resources among many users. Users typically do not have a choice of batch systems to use on a particular resource - they simply use the interfaces provided by the batch systems that are made available on those resources.

The MapReduce programming model [17], introduced by Google, has become popular over the past few years as an alternative model for data parallel programming. Apart from Google's proprietary implementation of MapReduce, there are several popular open source implementations available such as Apache Hadoop MapReduce [11] and Disco [16]. MapReduce technologies have also been adopted by a growing number of groups in industry (e.g., Facebook [27], and Yahoo [21]). In academia, researchers are exploring the use of these paradigms for scientific computing, for example, through the Cluster Exploratory (CluE) program, funded by the National Science Foundation (NSF).

A growing number of codes in scientific domains such as Bioinformatics ([24], [20]) and Geosciences [22] are being written using open source MapReduce tools such as Apache Hadoop. In the past, these users have had a hard time running their Hadoop codes on traditional HPC systems that they have access to. This is because it has proven hard for Hadoop to co-exist with existing HPC resource management systems, since Hadoop provides its own scheduling, and manages its own job and task submissions, and tracking. Since both systems are designed to have complete control over the resources that they manage, it is a challenge to enable Hadoop to co-exist with traditional batch systems such that users may run Hadoop jobs on these resources. Furthermore, Hadoop uses a *shared-nothing* architecture [30], whereas traditional HPC resources typically use a shared-disk architecture, with the help of high performance parallel file systems. Due to these challenges, HPC users have been left with no option other than to procure a physical clus-

ter and manage and maintain their own Hadoop instances. Some users now have access to new resources such as Amazon's Elastic MapReduce [1] or Magellan [3] to run their Hadoop jobs. However, the majority of HPC users only have access to traditional HPC-style resources, such as the ones provided by the TeraGrid or other local supercomputing facilities.

In this paper, we present *myHadoop*, which is a simple framework for Hadoop *on-demand* on traditional HPC resources, using standard batch processing systems such as TORQUE or SGE. With the help of myHadoop, users do not need dedicated clusters to run their jobs - instead, they can configure Hadoop clusters on-demand by requesting resources via TORQUE or SGE, and then configuring the Hadoop environment based on the set of resources provided. We describe the architecture of myHadoop, and evaluate the performance overhead of using such a system with a few scientific use-case scenarios. myHadoop is open source, and available for download via SourceForge [4].

The key contributions of our work are as follows:

(i) An open-source framework for leveraging traditional batch systems to run Hadoop jobs on HPC resources,

(ii) A detailed recipe for implementing a "shared-nothing" system such as Hadoop on shared HPC resources, which may be useful for other similar systems, e.g. Disco [16], and

(iii) An evaluation of the performance overheads of running a shared-nothing infrastructure on such resources.

The rest of the paper is organized as follows. In Section 2, we describe the traditional shared HPC architectures, and shared-nothing architectures used by Apache Hadoop. We discuss the challenges of running MapReduce-style applications on shared HPC resources. In Section 3, we discuss the myHadoop implementation details. In Section 4, we evaluate the performance implications of using myHadoop with the help of two use cases. We present related work in Section 5, and our conclusions and future work in Section 6.

2. ARCHITECTURE OVERVIEW

The system architecture for shared-nothing frameworks such as Apache Hadoop is different from that of the traditional shared HPC resources, as shown in Figure 1.

We observe that most HPC resources are composed of a set of powerful compute nodes with minimal local storage (e.g. [8], [5], etc). The compute nodes are themselves connected to each other using a high-speed network interconnect such as Gigabit Ethernet, Myrinet [13] or Infiniband [26]. They are typically connected to a high performance parallel file system, such as Lustre [29] or IBM's General Parallel File System (GPFS - [28]). Access to the compute nodes is via batch systems such as TORQUE/PBS or SGE.

Shared-nothing frameworks, on the other hand, are designed for use in large clusters of commodity PCs connected together with switched commodity networking hardware, with storage directly attached to the individual machines. Thus, every machine is both a data and a compute node. A dis-
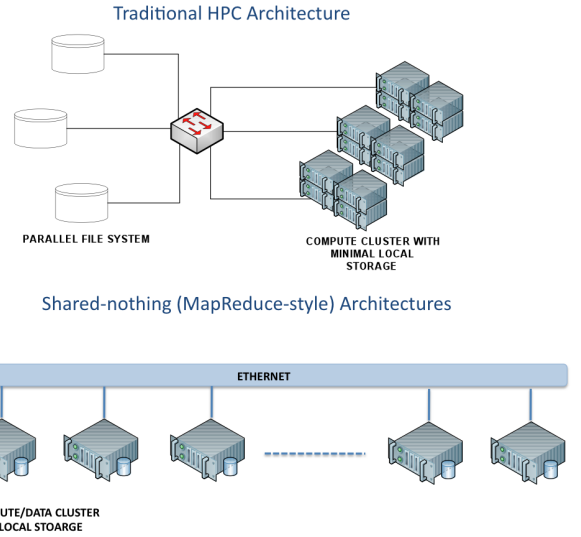

Figure 1: HPC versus shared-nothing architectures

tributed file system is implemented on top of the data nodes. e.g. the Google File System (GFS - [19]), or the Hadoop Distributed File System (HDFS - [14]). Compute tasks are spawned to maximize data locality. Such systems have been documented to scale to thousands of nodes effectively ([17], [27], [21]).

The main challenges in enabling Hadoop jobs to run on shared HPC resources are:

(i) Enabling the resource management functions of Hadoop to co-exist with the native batch resource managers in some fashion, and

(ii) Implementing a shared-nothing infrastructure on top of the traditional HPC architectures, which are typically shared-disk systems.

Hadoop uses a shared-nothing architecture, but it designates one node as the *master*, and the rest as the *slaves*. On the master, Hadoop runs the HDFS *NameNode* daemon, which is the master server that manages the file system namespace and regulates access to files by clients, and the *JobTracker* daemon, which is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slave nodes host the HDFS *DataNode* daemons, which manage storage attached to the nodes, and the MapReduce *TaskTracker* daemons, which execute the tasks as directed by the master.

Figure 2 shows the overall architecture of myHadoop on traditional HPC resources. In summary, a Hadoop cluster is allocated on-demand by first requesting a set of nodes from the native resource management system, designating the master and the slave nodes, and configuring and launching the appropriate Hadoop daemons on the allocated nodes. The user can then run their Hadoop jobs, after which the Hadoop framework is torn down by stopping all the daemons and de-allocating the resources.
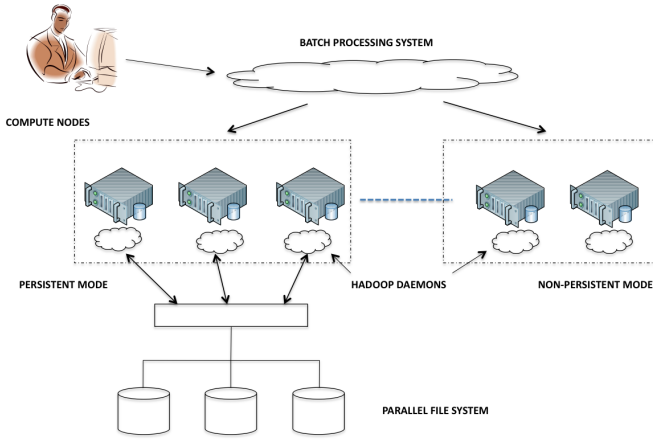
Figure 2: myHadoop architecture

Figure 3: myHadoop configuration workflow

myHadoop can be configured in two modes - *non-persistent* and *persistent*. In the non-persistent mode, the Hadoop daemons are configured to use local storage, if available, for the distributed file system implementation. This mode may have two potential problems - first, sufficient local storage may not be available, and second, the results from the non-persistent runs will be unavailable after the Hadoop job has completed, since the batch system cannot typically guarantee that the same set of resources will be allocated for future runs. To circumvent these concerns, one can use the persistent mode where the distributed file system is hosted on the shared file system, such as Lustre or GPFS.

3. IMPLEMENTATION DETAILS
The requirements for myHadoop can be listed as follows:

(i) Enabling execution of Hadoop jobs on shared HPC resources via traditional batch processing systems,

(ii) Working with a variety of batch scheduling systems,

(iii) Allowing users to run Hadoop jobs without needing root-level access,

(iv) Enabling multiple users to simultaneously execute Hadoop jobs on the shared resource (this doesn't imply that they should use the same Hadoop instance - only that the Hadoop configurations for one user must not interfere with the configuration of another), and

(v) Allowing users to either run a fresh Hadoop instance each time (non-persistent mode), or store HDFS state for future runs (persistent mode).

myHadoop has been implemented to work with Apache Hadoop (version 0.20.2), and to satisfy the above requirements. The key idea behind the implementation is that different (site-specific) configurations for Hadoop can be generated for different users, which can then be used by the users to run personal instances of Hadoop in regular-user mode, without needing any system-wide configuration changes or root privileges. Site-specific configuration files that are relevant
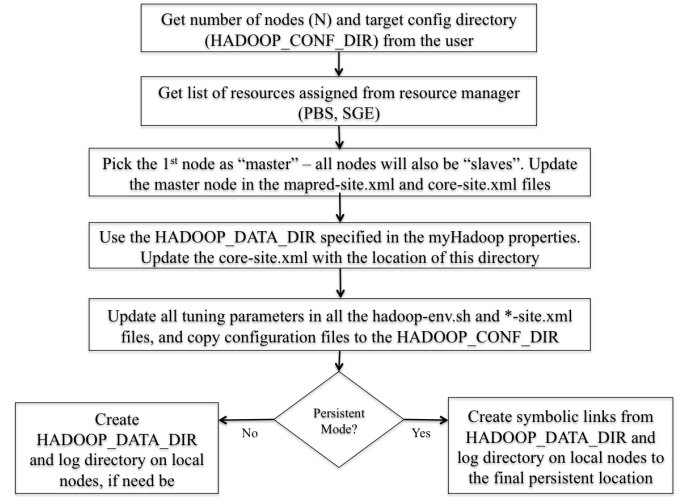
to myHadoop include:

(i) *masters*: This specifies the host name of the node on the cluster that serves as the master. This node hosts the HDFS *NameNode*, and the MapReduce *JobTracker* daemons.

(ii) *slaves*: This lists the host names for the compute nodes on the cluster. The slave nodes host the HDFS *DataNode* daemons, and the MapReduce *TaskTracker* daemons.

(iii) *core-site.xml*: The core site configuration includes important parameters such as the location of the HDFS (*HADOOP_DATA_DIR*) on every node, and the URI for the HDFS server (which includes the host and port of the master). It also includes additional tuning parameters for the size of the read/write buffers, the size of the in-memory file system used to merge map outputs, and the memory limit used for sorting data.

(iv) *hdfs-site.xml*: The HDFS site configuration includes parameters for configuring the distributed file system, such as the number of replications, the HDFS block size, and the number of DataNode handlers to serve block requests.

(v) *mapred-site.xml*: The MapReduce site configuration consists of the host and port for the JobTracker (on the master), the number of parallel copies that can be run by the Reducers, the number of map and reduce tasks to run simultaneously (to leverage multiple cores), and the *JAVA_OPTS* for the child JVMs of the mappers and reducers.

(vi) *hadoop-env.sh*: This script configures the environment for the Hadoop daemons. Important parameters including the location of the log directory, the Hadoop heap size, and JVM parameters for garbage collection and heap management.

Figure 3 describes the myHadoop workflow for configuration of a Hadoop cluster on-demand for every user. To use myHadoop, a user writes scripts for the batch system

being used by their particular resource. For the purposes of this discussion, let us assume that the resource uses the TORQUE Resource Manager (also known as PBS). Note that the following description is equally valid for other resource managers, such as SGE.

In this case, a user writes a regular PBS script to run their Hadoop job. From within the PBS script, the user invokes myHadoop scripts for configuration of a Hadoop cluster. When the Hadoop configuration script is invoked, it sets up all the configuration files for a personal Hadoop instance for a user in a separate directory (called *HADOOP_CONF_DIR*), which can then be used to bootstrap all the Hadoop daemons. As command-line arguments to this script, the user passes the number of Hadoop nodes to configure (which is the same as the number of nodes requested from PBS), the HADOOP_CONF_DIR to generate the configuration files in, and whether Hadoop should be configured in persistent or non-persistent mode. When this script is invoked, my-Hadoop looks up the host names of the resources allocated to it by PBS, using the *PBS_NODEFILE*. It picks the first resource on the list as the master, and all of the resources as slaves. It updates the masters and slaves files accordingly, and also the mapred-site.xml and core-site.xml, which contain the host names for the HDFS NameNode and MapReduce JobTracker respectively. It then reads the location of the HADOOP_DATA_DIR from its set of pre-defined properties, and updates the core-site.xml. It then updates all the tuning parameters based on the site specific configuration files, and writes out all the relevant configuration files into the HADOOP_CONF_DIR.

If a user wants to run Hadoop in regular (or non-persistent) mode, then myHadoop creates the HADOOP_DATA_DIR on all the nodes, and HDFS can then be formatted. If a user wants to run Hadoop in persistent mode, then myHadoop creates symbolic links from the HADOOP_DATA_DIR on each individual node to the location on the shared file system to be used to host the HDFS. For instance, a symbolic link is created from HADOOP_DATA_DIR to *BASE_DIR/$i* for every compute node *$i*.

The myHadoop workflow from a user's perspective is shown in Figure 4. As described above, a user writes a PBS script to request the required number of resources. Then the user configures the site-specific parameters using the myHadoop configuration scripts. Then, using the configuration files generated in the HADOOP_CONF_DIR, the user formats HDFS (optional in persistent mode, mandatory in non-persistent mode) and starts the Hadoop daemons. The user then stages the required input files into HDFS using Hadoop commands, and is now ready to run her Hadoop jobs. Once the Hadoop jobs are finished, the results can be staged back out from HDFS. This step is necessary in the non-persistent mode, because the output files are distributed across the compute nodes, and there is no guarantee that this user will be allocated the exact same set of nodes in the future by PBS. Thus, all results must be staged out before the resources are de-allocated. However, this step is not necessary in the persistent mode since the results will be available on the shared file system even after the PBS job has completed. Finally, the user shuts down all Hadoop daemons and exits from PBS.
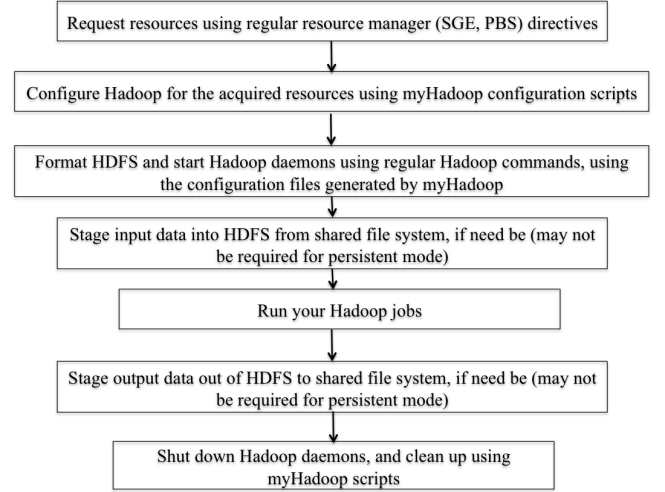

Figure 4: myHadoop from a user's perspective

Thus, myHadoop enables running Hadoop jobs on HPC resources using standard batch processing systems. It is possible that a similar approach could be used to implement other shared-nothing frameworks, such as Disco [16], on traditional HPC resources.

4. PERFORMANCE EVALUATION

As discussed before, myHadoop configures and bootstraps execution of Hadoop daemons prior to the execution of Hadoop jobs, and performs cleanup after job execution is complete. Hence, there is certainly some overhead involved in the overall execution time of the Hadoop job. The overheads are different for the persistent and non-persistent modes.

For the non-persistent mode, the overheads include the configuration (generation of site specific configuration files, creation of HDFS data directories, etc), staging input data into HDFS, staging the results back to persistent storage, and shutting down and cleaning up all the Hadoop daemons. For the non-persistent mode, configuration and shutdown are the only true overheads. Assuming that the data are already stored and persisted in HDFS, then the initial load times can be amortized over the overall lifetime of the project. Exporting data from HDFS to a regular Unix file system may sometimes be necessary even in persistent mode, in the cases where the results need to be shared with other non-Hadoop applications. In this case, staging outputs from HDFS may be considered as an overhead. Using two examples, we evaluate the performance of myHadoop and its associated overheads.

Note that the purpose of our experiments is just to measure the performance implications of using myHadoop, and not to extensively study the performance characteristics of the applications themselves. To help illustrate the effects of running shared-nothing Hadoop jobs on typical shared HPC resources, we have chosen two classes of applications, (i) Hadoop-Blast ([6], [20]), which is compute-intensive and uses only a modest amount of data, and (ii) a Hadoop-based

core-site.xml:

io.file.buffer.size	131072	Size of read/write buffer
fs.inmemory.size.mb	650	Size of in-memory FS for merging outputs
io.sort.mb	650	Memory limit for sorting data

hdfs-site.xml:

dfs.replication	2	Number of times data is replicated
dfs.block.size	134217728	HDFS block size in bytes
dfs.datanode.handler.count	64	Number of handlers to serve block requests

hdfs-site.xml:

mapred.reduce.parallel.copies	4	Number of parallel copies run by reducers
mapred.tasktracker.map.tasks.maximum	4	Max map tasks to run simultaneously
mapred.tasktracker.reduce.tasks.maximum	2	Max reduce tasks to run simultaneously
mapred.job.reuse.jvm.num.tasks	1	Reuse the JVM between tasks
mapred.child.java.opts	-Xmx1024m	Large heap size for child JVMs

Figure 5: List of Hadoop tuning parameters

sub-selection of high-resolution topographic data sets from the OpenTopography project ([23]), which is highly data-intensive.

4.1 Environment Overview

All of our experiments were run on the *Triton Compute Cluster (TCC)* at the San Diego Supercomputer Center. The cluster features Appro gB222X Blade Server nodes with dual quad-core Intel Xeon E5530 processors with Intel Microarchitecture Nehalem, running at 2.40 GHz. Each of the 256 nodes has 24 gigabytes of memory and an eight-megabyte cache. Each node has a 10-gigabit Myrinet connection, giving the system a total bandwidth capacity of 256 gigabytes per second. Triton uses the TORQUE Resource Manager, with the Moab Workload Manager to manage job queues.

We use Apache Hadoop version 0.20.2 for our experiments. The tuning parameters used for our experiments are shown in Figure 5. These are pre-configured for the cluster by the system administrators via myHadoop. Users can optionally update the parameters for their runs - however, it is not recommended that the users update their configurations unless they are extremely familiar with Hadoop administration. In general, we have found that query performance improves with replication (e.g. with $dfs\_replication = 2$), with only a minimal penalty during data load for the non-persistent mode.

For the persistent mode, we use the Triton *Data Oasis* resource, which is designed as an extremely large scale storage system, having two-to-four petabytes of total disk capacity when fully deployed. The current environment includes a set of Lustre-based Parallel File System [29] resources supporting about 250 terabytes of usable storage space. Data Oasis is connected to the cluster nodes via 10-gigabit Ethernet links. For the non-persistent mode, we use local disk on the individual cluster nodes to host HDFS.

4.2 Hadoop-Blast

The Basic Local Alignment Search Tool (BLAST) is a popular Bioinformatics family of programs that finds regions of
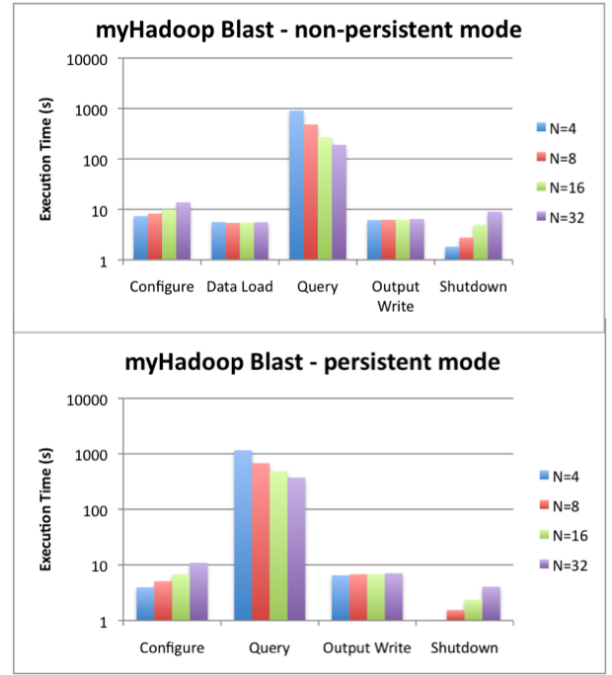
Figure 6: Hadoop-Blast using myHadoop

local similarity between sequences [10]. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families. Hadoop-Blast [6] is a MapReduce-based tool which lets a user run the Blast programs to compare a set of input query sequences against standard databases. The implementation is quite straightforward - for every input file (query sequence), Hadoop-Blast spawns a new map task to execute the appropriate Blast program, with the user specified parameters. An output file is created for each map task with the results of the Blast run. There is no need for a reduce task in this workflow.

For our experiments, we run *blastx*, which is one of the available Blast programs that compares the six-frame conceptual translation products of a nucleotide query sequence (both strands) against a protein sequence database. As inputs, we use 128 query sequences of equal length (around 70K each), which are compared against the *nr* peptide sequence database (around 200MB in size). Figure 6 shows the performance for the various steps of the execution. The y-axis (*Execution Time*) is in log-scale.

As seen in the performance graphs, the time required to configure a Hadoop cluster using myHadoop configuration scripts is between 8 to 15 seconds for clusters varying from *N=4* to *32* nodes. In the non-persistent mode, all the input sequences and the reference database need to be staged in - this step is not necessary for the persistent mode. However, since the amount of data to be staged is in the order of a few hundreds of MB, data stage-in accounts for an overhead of less than 10 seconds. The blastx runs themselves are

computationally intensive, and scale pretty well with the number of nodes. The total execution time varies from a few hundreds of seconds for 32 nodes, to around 1000s for 4 nodes. Staging outputs takes less than 10 seconds, because of the modest amount of data to be staged out. Shutdowns are also of the order of a few seconds. In summary, the overheads are minimal for fewer nodes (around 2% for 4 nodes in the non-persistent mode). They are greater for larger number of nodes (around 20% for 32 nodes in non-persistent mode), however, since increasing the number of nodes greatly reduces the query time, the overall execution time is greatly reduced in spite of the increased overhead in configuring, bootstrapping, and tearing down the Hadoop cluster.

Other interesting observations from the graphs are as follows. The query execution time for the non-persistent mode, which uses local disk, is observed to be faster than the persistent mode, which uses the Lustre file system. We believe that this is because Lustre is designed for large sequential IO workloads, and not for providing access to a large number of smaller files. Since the data sizes are not very large, the use of Lustre to host the HDFS causes a drop in the performance for this experiment. Next, the configuration and shutdown process for the persistent version is a few seconds faster than the non-persistent version. This is because the HDFS doesn't have to be formatted during configuration, and all the HDFS data doesn't have to be cleaned up in the shutdown process for this mode. However, the difference is an insignificant percentage of the overall execution time.

In summary, Hadoop-Blast is an excellent candidate for the use of myHadoop in non-persistent mode. The use of persistent mode does not provide any greater benefit - in fact, it only acts as a detriment. The key characteristics of Hadoop-Blast that cause this behavior are that it is a compute-intensive application that is embarrassingly parallel, and it deals with only a limited amount of data - from hundreds of megabytes to a few gigabytes. Other applications with similar characteristics are also best served by using myHadoop in non-persistent mode.

4.3 Hadoop Data Sub-selection

LIDAR (Light Detection and Ranging) is a remote sensing technology that combines a high-pulse rate scanning laser with a differential global positioning system (GPS), and a high-precision inertial measurement instrument on an aircraft to record dense measurements of the position of the ground, overlying vegetation, and built features. Firing up to several hundred thousand pulses per second, LIDAR instruments can acquire multiple measurements of the Earth's surface per square meter over thousands of square kilometers. The resulting data set, a collection of measurements in geo-referenced X, Y, Z coordinate space known as a "point cloud", provides a 3- dimensional representation of natural and anthropogenic features at fine resolution over large spatial extents. The OpenTopography facility at SDSC provides online access to terabytes of such data, along with processing tools, and other derivative products.

The initial step in every LIDAR workflow is the sub-selection of points given a bounding box that covers a region of interest. Once the data is selected, users typically compute
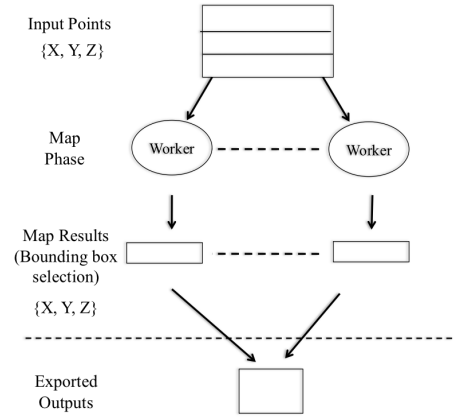

Figure 7: Hadoop implementation of bounding box-based point cloud sub-selection

Digital Elevation Models (DEM) to generate a digital continuous representation of the landscape. The DEMs can then be used for a range of scientific and engineering applications, including hydrological modeling, terrain analysis, and infrastructure design. We are investigating the use of MapReduce technologies to implement the entire workflow [22]. For this experiment, we focus on the initial step of the workflow, which is bounding box-based point cloud sub-selection. Figure 7 shows the MapReduce implementation of the bounding box sub-selection using Hadoop. The sub-selection is done in the map phase - no reduce tasks are necessary.

Figure 8 shows the performance of the data sub-selection using myHadoop in both the non-persistent and persistent modes, for data sizes from 1GB to 100GB, using 4 and 32 nodes. For all the runs, we are running a bounding box query that selects around 12.5% of the loaded data. It can be observed that data loads dominate the the execution time for the non-persistent mode. In fact, the time taken for loading data into HDFS from Lustre is observed to be an order of magnitude greater than the queries themselves. Furthermore, staging results back is also time consuming. For the persistent mode, data is staged in once, and the cost is amortized over time - hence, we do not show the load time on our graphs. Staging of outputs is required in the persistent mode only if the data sub-selection is the final step of the workflow. In practice, this is often not the case - selected data is typically fed into the next step of the workflow, which is the DEM generation. However, for the sake of completeness, we have also shown the output staging times on the graph.

An interesting observation from Figure 8 is that the query time is actually faster in the persistent mode, which is opposite of what we found in Section 4.2. This is consistent with our theory that Lustre is optimized for large sequential IO, and that our data sizes are significantly larger in this experiment. Greater performance improvement is observed for 32 nodes for the larger data sizes, whereas the performance improvement is not as significant for the smaller data sizes. This is to be expected since 4 nodes, tuned as per the settings in Figure 5, are sufficient to handle the smaller queries.
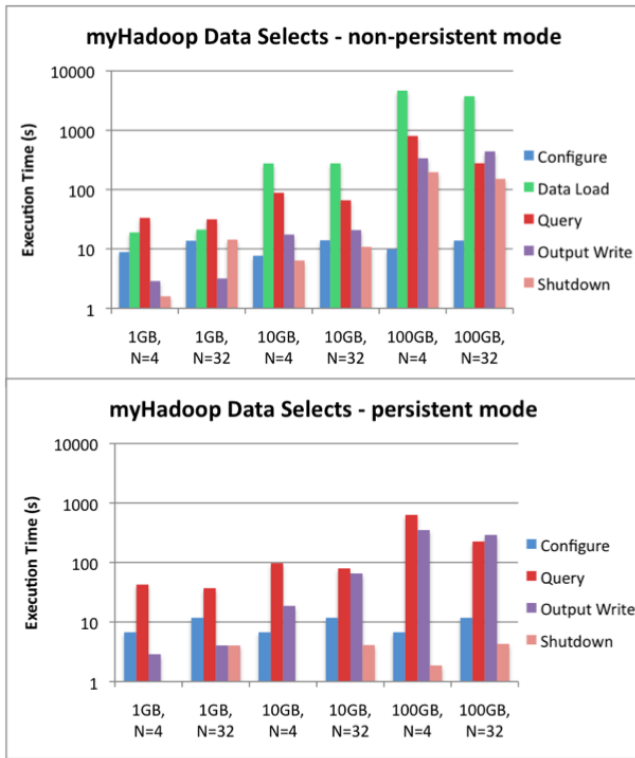
Figure 8: Bounding box selection using myHadoop

In summary, the persistent mode of myHadoop may be a good candidate for large data-intensive applications, such as the usage scenario described above. For such applications, the non-persistent mode of myHadoop provides significantly worse performance. In the non-persistent mode, most of the time is spent staging data in and out of Hadoop for such use cases. The persistent mode of myHadoop is especially suitable for workflows where results from one stage are fed to subsequent stages, thus minimizing the staging of data back and forth from HDFS to the native file systems provided on the resources.

5. RELATED WORK

There has been a lot of user interest in recent times for running Hadoop jobs on traditional HPC resources. Several groups have worked on ad-hoc scripts to solve this problem. However, there are very few efforts that have been sufficiently documented, and publicly available. myHadoop is very similar in concept to the description provided by the blog article [18], where the author describes the process of getting Hadoop to run as a batch job using PBS. However, myHadoop is a more general and configurable framework, which provides support for other schedulers as well, and is freely available for download via SourceForge. Furthermore, to our knowledge, there has not been a performance evaluation of this effort that has been published.

The Apache Hadoop on Demand (HOD - [2]) is a system for provisioning virtual Hadoop clusters over a large physical cluster. It uses the TORQUE resource manager to do node allocation. On the allocated nodes, it can start Hadoop

Map/Reduce and HDFS daemons. It automatically generates the appropriate configuration files for the Hadoop daemons and client. HOD also has the capability to distribute Hadoop to the nodes in the virtual cluster that it allocates. HOD differs from myHadoop in the following ways. Firstly, HOD requires the use of an external HDFS that is statically configured. This means that the MapReduce jobs can't exploit any data locality, because the data is not co-located with the map and reduce tasks. The non-persistent mode of myHadoop enables data locality, with the HDFS being dynamically configured to use the same nodes, with the caveat that the data does have to be staged in and out before and after the execution. The HDFS implementation in the persistent mode of myHadoop is similar in concept to the statically configured external HDFS used by HOD. However, in the persistent mode, the HDFS in myHadoop may be thought of as pseudo-local because there is a 1-1 mapping between every node that runs the MapReduce daemons and its corresponding external HDFS data directory. Another difference is that HOD is based on the TORQUE resource manager - while myHadoop is not limited to just TORQUE. Finally, HOD has some documented problems ([18]) in setting up multiple concurrent Hadoop instances simultaneously - however, it does enable sharing of an on-demand Hadoop instance between users. myHadoop is designed to support concurrent personal instances of Hadoop for multiple users.

CloudBatch [31] is a batch job queueing system on clouds, which uses Hadoop to manage both regular and MapReduce computing needs. For example, it can enable a user to run PBS jobs via Hadoop. This implies that all the resources use Hadoop as their native scheduling systems, which may not be a feasible option for most computing facilities. Finally, Amazon's Elastic MapReduce [1] enables the allocation of Hadoop clusters on the fly using a Web service API on Amazon's cloud resources. It is similar to myHadoop in the sense that Hadoop clusters are launched on-demand, and that it uses an external location for data persistence (Amazon's Simple Storage Service - S3). However, the difference lies in the fact that it uses Amazon's cloud resources, rather traditional HPC resources via batch scheduling systems.

6. CONCLUSIONS & FUTURE WORK

In this paper, we described myHadoop, a framework for configuring Hadoop on-demand on traditional HPC resources, using standard batch scheduling systems such as TORQUE (PBS) or the Sun Grid Engine (SGE). With the help of myHadoop, users with pre-existing Hadoop codes do not need dedicated Hadoop clusters to run their jobs. Instead, they can leverage traditional HPC resources that they otherwise have access to, without needing root-level access. myHadoop enables multiple users to simultaneously execute Hadoop jobs on shared resources without interfering with each other. 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. In this paper, we discussed the performance of both the persistent and non-persistent modes with the help of a few usage scenarios, and provided recommendations on when myHadoop would be a suitable option (or otherwise). myHadoop is open source

and freely available for download via SourceForge [4].

The current release of myHadoop is early alpha, and several potential improvements remain to be done. In particular, we are planning on adding support for other schedulers such as Condor [12]. Currently, the support for the persistent mode is quite basic. In particular, one shortcoming of the persistent mode is that a user can only instantiate a new Hadoop instance in the future with the same number of nodes as their first instance that was initialized, if the user wants to re-use any data from previous runs. A desirable feature is to be able to dynamically re-configure the number of nodes, and also re-balance the data between the nodes. Another shortcoming is that the data in the shared persistent location is only accessible via HDFS commands - which implies that a user must instantiate a Hadoop cluster via myHadoop if any data needs to be exported on to the native file system. We are planning on implementing a set of command-line utilities that help write and read data to and from the persistent location being used by myHadoop's HDFS.

7. ACKNOWLEDGEMENTS

8. REFERENCES

[1] Amazon Elastic MapReduce. 2011. http://aws.amazon.com/elasticmapreduce/.

[2] Apache Hadoop on Demand (HOD). 2011. http://hadoop.apache.org/common/docs/r0.20.2/hod_user_guide.html.

[3] Magellan: NERSC Cloud Testbed. 2011. http://www.nersc.gov/nusers/systems/magellan/.

[4] myHadoop on SourceForge. 2011. http://sourceforge.net/projects/myhadoop/.

[5] NCSA Abe Cluster Technical Summary. 2011. http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/Intel64Cluster/TechSummary/.

[6] Running Hadoop-Blast in Distributed Hadoop. 2011. http://salsahpc.indiana.edu/tutorial/hadoopblastex3.html.

[7] The Sun Grid Engine (SGE), 2011. http://wikis.sun.com/display/GridEngine/Home.

[8] The Triton Compute Cluster. 2011. http://tritonresource.sdsc.edu/cluster.php.

[9] TORQUE Resource Manager, 2011. http://www.clusterresources.com/products/torque-resource-manager.php.

[10] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[11] Apache Software Foundation. Hadoop MapReduce. 2011. http://hadoop.apache.org/mapreduce.

[12] J. Basney, M. Livny, and T. Tannenbaum. High Throughput Computing with Condor. In *HPCU news*, volume 1(2), June 1997.

[13] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A gigabit-per-second local area network. *Micro, IEEE*, 15(1):29–36, 2002.

[14] D. Borthakur. The hadoop distributed file system: Architecture and design, 2007. *Apache Software Foundation*.

[15] C. Catlett. The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility. In *2nd IEEE/ACM Intl Symp on Clust Comp & the Grid*, 2005.

[16] N. R. Center. Disco MapReduce Framework. 2011. http://discoproject.org.

[17] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04: 6th Symp on Operating System Design and Impl*, 2004.

[18] J. Ekanayake. Hadoop as a Batch Job using PBS. 2008. http://jaliyacgl.blogspot.com/2008/08/hadoop-as-batch-job-using-pbs.html.

[19] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. *ACM SIGOPS Operating Sys Rev*, 37(5):29–43, 2003.

[20] T. Gunarathne, T. Wu, J. Qiu, and G. Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *19th ACM Intl Symp on High Perf Dist Comp*, pages 460–469. ACM, 2010.

[21] Y. Inc. Hadoop at Yahoo! 2011. http://developer.yahoo.com/hadoop.

[22] S. Krishnan, C. Baru, and C. Crosby. Evaluation of MapReduce for Gridding LIDAR Data. In *2nd IEEE Intl Conf on Cloud Comp Tech and Science*, 2010.

[23] S. Krishnan, V. Nandigam, C. Crosby, M. Phan, C. Cowart, C. Baru, and R. Arrowsmith. OpenTopography: A Services Oriented Architecture for Community Access to LIDAR Topography. SDSC TR-2011-1, San Diego Supercomputer Center, 2011.

[24] B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg. Searching for SNPs with cloud computing. *Genome Biol*, 10(11):R134, 2009.

[25] P. Papadopoulos, M. Katz, and G. Bruno. NPACI Rocks: Tools and techniques for easily deploying manageable linux clusters. In *cluster*, page 258. IEEE Computer Society, 2001.

[26] G. Pfister. An introduction to the InfiniBand architecture. *High Perf Mass Storage and Parallel I/O*, pages 617–632, 2001.

[27] J. S. Sarma. Hadoop - Facebook Engg. Note. 2011. http://www.facebook.com/note.php?note_id=16121578919.

[28] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *1st USENIX Conf on File and Storage Tech*, pages 231–244, 2002.

[29] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *2003 Linux Symp*, 2003.

[30] M. Stonebraker. The case for shared nothing. *Database Engineering Bulletin*, 9(1):4–9, 1986.

[31] C. Zhang and H. De Sterk. CloudBATCH: A Batch Job Queuing System on Clouds with Hadoop and HBase. In *2nd IEEE Intl Conf on Cloud Comp Tech and Science*, 2010.