The Cray Framework for Hadoop for the Cray XC30

Jonathan Sparks, Howard Pritchard, and Martha Dumler

Abstract—This paper describes the Cray Framework for Hadoop on Cray XC30. This is a framework for supporting components of the Hadoop eco-system on XC30's managed by widely used batch schedulers. The paper further describes experiences encountered in running Hadoop workloads over typical Lustre deployments on the Cray XC30. Related work to enable Hadoop to better utilize the XC high speed interconnect is discussed.

Index Terms—Hadoop, Cray XC30

1 Introduction

Hadoop is an open source, scalable software eco–system that can be used to solve many of the problems encountered by organizations trying to derive value from large amounts of data, whether structured, semi–structured, or unstructured [2]. Although Hadoop has gained wide acceptance in the data–intensive commercial space, and a few areas of HPC computing that map readily to the MapReduce programming model [8] (e.g. [12]), its adoption in the larger HPC community has been somewhat more limited.

There are a number of obstacles slowing the adoption of Hadoop as part of typical HPC workflows. One of the most significant issues involves workload management. Many Hadoop components (e.g. HBase [10], Accumulo [1]) are intended to be used as long running, shared services and are not well suited to the batch oriented model typically used for managing large HPC systems. Even for potentially more batch oriented components of Hadoop like MapReduce, the YARN resource manager/job launcher [9] used in Hadoop 2 does not interoperate well with typical HPC batch schedulers. Although efforts have been made to, for example, integrate Slurm with some components of Hadoop [6], [7], such efforts have failed to gain acceptance within the wider Hadoop community.

Another significant obstacle to broader adoption of Hadoop within the HPC community is the relatively tight coupling of Hadoop components to the Hadoop Distributed File System (HDFS), or more precisely, to the Hadoop Distributed File System Java class. Also, Hadoop works best on systems where each node has some amount of local persistent storage. Since Hadoop code is mainly developed and used where HDFS is deployed and nodes have local storage, deployment on typical HPC systems having no node-local storage, and with parallel file systems designed for HPC style I/O patterns can present some challenges, particularly at scale.

The Cray Framework for Hadoop for Cray XC30 is intended to address both the workload management challenges presented by the Hadoop ecosystem when used in a typical HPC environment, as well as address to some extent issues that arise from trying to use Hadoop on Lustre.

The rest of this paper is organized as follows. First, an overview of the Cray Framework for Hadoop is presented. This includes a high level description of to how install a Hadoop distribution and the Framework on a Cray XC 30 system. Various tuning options available to users are also discussed. A subsequent section describes experiences using Hadoop MapReduce at scale on a XC 30 system, and measures that can be taken to improve the performance of the application when using Lustre. Additional work Cray has undertaken to improve the performance of Hadoop components is also described.

2 Installing Cray Framework for Hadoop

The Cray Framework for Hadoop is based on the myHadoop project [3]. The Framework consists of Cray myHadoop – a set of scripts which enable a user of a Cray XC30 system managed either by PBS Pro or Moab/Torque to run Hadoop MapReduce jobs. Included are a default set of Hadoop configuration files tuned for typical Cray XC30 compute node configurations. The Hadoop eco-system components Pig, Mahout, and Crunch are also supported by the Framework. Also included in the package is a sample batch script users can modify for their particular Hadoop applications.

Cray myHadoop does not include a Hadoop distribution. A site wishing to make use of the Cray myHadoop must first install a Hadoop distribution in the shared root. Cray myHadoop has been tested with Apache Bigtop, Cloudera, HortonWorks, IDH (Intel Hadoop)

• *The authors are with Cray, Inc.*
E-mail: jsparks,howardp,mbd@cray.com

distributions. Cray myHadoop requires that the Apache Core Hadoop, Pig, Zookeeper, and Mahout be installed. Zookeeper is only needed for resolving RPM dependencies and support libraries. Note that all Hadoop services must be turned off in the default class view of the XC30 root filesystem. In addition, the *nc* binary (available in the netcat-openbsd package) is also required. The system must be running CLE 5.2 UP00 and also have CCM (Cray cluster compatibility package) installed.

After installing a Hadoop distribution, the Cray myHadoop rpm can be installed in the shared root. Before using Hadoop and the Cray myHadoop, some site specific customizations are recommended:

- A consistent version of the Java JVM needs to be used. The */etc/hadoop/conf/hadoop-env.sh* script needs to be modified so that JAVA_HOME points to the default Cray CLE JDK;
- Adjust the following YARN parameters to reflect the number of cpus per node (cores) and memory per node for the nodes in the XC system to be used for running Hadoop jobs: *yarn.nodemanager.resource.cpu-cores* and *yarn.nodemanager.resource.memory-mb*. These are in the *yarn-site.xml* configuration file that comes on the Cray myHadoop RPM.
- The system administrator may also wish to adjust default the memory limits for MapReduce mapper and reducer tasks. These are controlled via the *mapreduce.map.memory.mb*, *mapreduce.map.java.opts*, *mapreduce.reduce.memory.mb*, and *mapreduce.reduce.java.opts*. These parameters are specified in the *mapred-site.xml* configuration file that comes on the Cray myHadoop RPM.

Note that Hadoop works best when all of the nodes to be used for myHadoop batch jobs have the same amount of memory and cores per node. See [4] for further details on setting default Hadoop parameters.

In addition to the Hadoop tuning parameters described above, there are additional parameters controlling the location of YARN application log directories and application work directories that need to be considered. By default Cray myHadoop uses *tmpfs* for these directories. This helps considerably in mitigating issues encountered with the Lustre MDS when running larger (more than about 16 nodes) myHadoop jobs on a Cray XC30. However, since the tmpfs on each node is limited, jobs requiring a lot of memory for map and reducer tasks can fail. Rather than set Lustre as the system–wide default file system for these directories, it is recommended that users running Hadoop jobs which cannot use tmpfs do the following:

- Make a copy of the contents of */opt/cray/myHadoop/default* to a directory (designated X in this example) where the user has read write permissions and which is mounted on the compute nodes.
- The user should modify the line in X/bin/functions.sh where the *yarnlocalstoredirtmp* is set.
- In the user's batch script set the *MYHADOOP_HOME* environment variable to point to directory X.

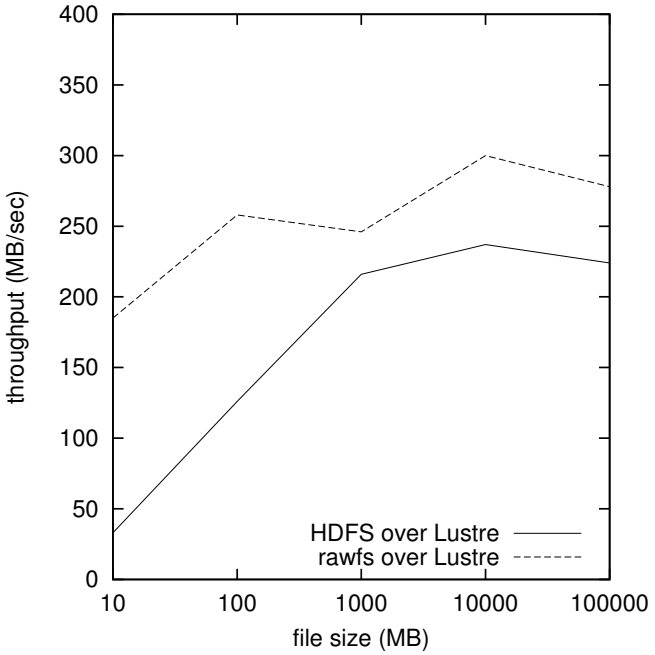

Fig. 1. Single node DFSIO Test using HDFS over Lustre and rawfs over Lustre

There are a number of environment variables users can employ in their myHadoop batch jobs to fine tune the behavior of Hadoop and the myHadoop framework. See Table 1.

3 Observations and Experiences using Cray myHadoop

A typical Hadoop deployment, in which many elements of the Hadoop eco–system are utilized, including Hbase, etc. makes extensive use of HDFS. Application jar files are copied to HDFS as part of job launch, YARN saves job logfiles to HDFS, etc. Input and output data from MapReduce jobs are stored on HDFS. Basically any persistent data is stored on HDFS.

In initial investigations of how to support Hadoop on Cray XC systems, it was considered important to try and support HDFS which meant running HDFS on top of Lustre. Although HDFS can be run on top of Lustre, there are significant performance penalties. Also as attempts were made to run larger jobs using HDFS over Lustre, numerous issues in the Lustre software stack were encountered.

Tests using the DFSIO benchmark show the impact of the HDFS protocol when running over Lustre verses just having the DFSIO benchmark read and write directly to Lustre. Figure 1 compares throughput rate obtained by the DFSIO test for a single process when reading/writing to HDFS over Lustre and directly to Lustre

TABLE 1
Environment variables for Cray myHadoop

Variable	Description	Default or Suggested Value
MYHADOOP_DEBUG	Turns on script debugging output, true or false	false
MYHADOOP_HOME	Location of myHadoop scripts	/opt/cray/myHadoop/default
HADOOP_PREFIX	Location of installed Hadoop User scripts will need to set to /alt-root if Hadoop is installed in an alternate root location	/
	/alt-root	
HADOOP_USER_PATH_FIRST	Prepend Hadoop paths to user path, true or false	false
HADOOP_USER_CLASSPATH_FIRST	Prepend Hadoop classpath to classpath, true or false	true
MYHADOOP_FSTYPE	Switches Hadoop to use the built in raw file system class, or HDFS. Values: hdfs or rawfs	rawfs

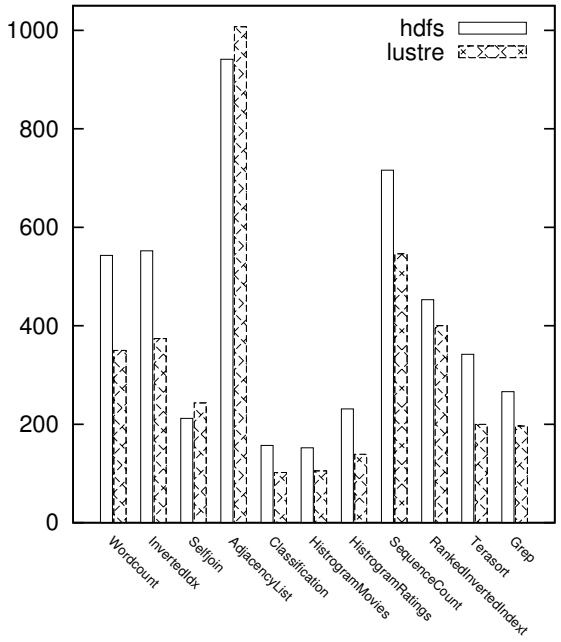

Fig. 2. Runtime for PUMA benchmarks using HDFS vs using Lustre (Hadoop raw file system class)

using the raw file system class available in Hadoop - the file system class Hadoop uses when writing to local file systems. Note that for the HDFS default block size (64MB), using the raw file system yields over a two–fold improvement in throughput rate compared to HDFS. In addition, as shown in Figure 2 using Lustre directly for job input and output, rather than HDFS, was found to speed up almost all of the PUMA Hadoop benchmarks [5] significantly. In addition, running HDFS over Lustre at scale – in terms of data stored – presents other challenges, particularly in the growth of file metadata, since HDFS adds yet another block storage construct on top of the striping notion intrinsic to the Lustre file system.

Fortunately, a refinement of efforts to support Hadoop on Cray XC30 systems allowed for the revisiting of the assumption that HDFS needed to be supported. If usage of Hadoop is restricted to a subset of the Hadoop eco–system, namely only supporting MapReduce, workflow management software for MapReduce workloads, and a few other applications making use of MapReduce, then the need to support HDFS on XC30 systems becomes optional. Note that without the presence of a global file system already (Lustre) this would not be the case.

Although turning off HDFS over Lustre helps, as attempts were made to scale up job sizes run within myHadoop, the ways in which Hadoop uses filesystems still resulted in quite a number of issues with Lustre. Lustre is intended primarily for use by HPC style applications, which tend to focus on reading and writing data to files. Although Hadoop does this, parts of Hadoop, for example the *OutputCommitter class*, tend to do a lot of meta-data type operations: getting file attributes, renaming files and directories, traversing directory structures, changing permissions on files and directories, etc. Depending on the Hadoop workload, many threads on a given node can be simultaneously issuing these types of meta-data operations. The Lustre meta-data server (MDS) is not designed to hold up very well under these types of I/O workloads when many nodes are being used for a Hadoop job.

To mitigate these Lustre issues, the default myHadoop configuration was changed to use local ramfs on the compute nodes for as many intermediate files (YARN task logs, application intermediate output files – such as map output files from mapper tasks) as possible. By using the compute node ramfs, many of the problems encountered with Lustre can be bypassed. Using ramfs rather than Lustre also speeds up many MapReduce jobs significantly.

The startup overhead of the myHadoop framework itself was also found to be a bottleneck when scaling beyond a few dozen nodes. By not using HDFS (and thus eliminating the startup associated with starting the

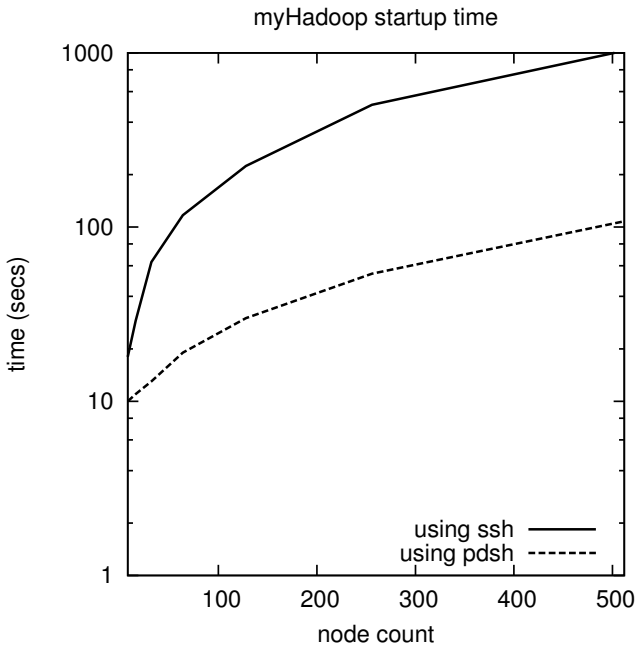

Fig. 3. Comparison of myHadoop startup times using ssh and pdsh.

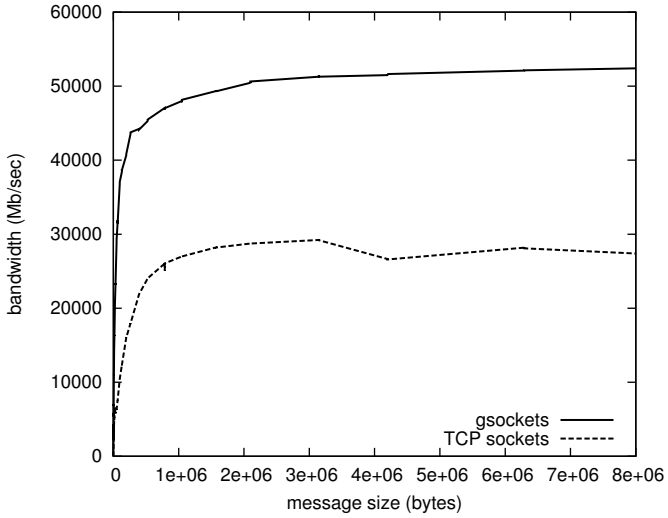

Fig. 4. Comparison of Bandwidth using gsockets vs TCP sockets on Cray XC30.

namenode and datanode daemons), the startup time was reduced somewhat, but not sufficiently to make the script practical for more than about 32 nodes. By replacing the use of ssh to startup the YARN nodemanagers with pdsh, a much more significant speedup was realized. Replacing ssh with pdsh yielded a speedup of about 20 when launching the framework across 512 nodes. See Figure 3.

4 Other Work

At first glance, when using MapReduce with a global shared file system like Lustre, it would seem an obvious place for improvement of the algorithm would be to bypass the *http get* method by which reducer tasks access the *map output* files where mapper tasks had previously executed. Note that *map output* files are written to *local* storage on a typical Hadoop cluster, not to HDFS. In contrast, on an XC30 system using Cray myHadoop, the *map output* files may optionally be written to Lustre rather than *ramfs*.

With Hadoop 2, MapReduce supports a plugin model which allows one to override the default MapReduce shuffle method with a different Java class. As part of the Cray Framework for Hadoop, a Shuffle class was implemented in which reducer tasks bypass the *http get* for obtaining the map output file data by reading the data directly from Lustre. It was found that this method actually is slower for cases where the map output files are small enough to remain within the file system buffer cache of the nodes where mapper tasks were run. It is actually faster in this case for the YARN nodemanager daemon (where the http server for handling the map output file transfers is run) to send the data back to the requesting reducer task directly – without having to retrieve the data from slow spinning media. In the course of analyzing the performance of the shuffle method it was found that on XC30 systems running MapReduce jobs of potential interest for HPC applications – where its typically one of distilling a large amount of input data into a small set of interesting data points – this is the more frequent MapReduce usage model. However, for large Extract-Transform-Load (ETL) style MapReduce jobs running at modest 4-8 node counts, the plugin may prove useful. This *Lustre–aware* shuffle plugin is included as part of the Cray Framework for Hadoop. Note that to use this plugin, the user must follow the steps described in section 2.

Another avenue was also explored for improving the performance of Hadoop, as well as other applications which make use of TCP–sockets for inter–node communication. Using the OFED rsockets package as a starting point, this user–space based sockets over IBverbs library was enhanced to use the XC30 native GNI api. This native GNI port of rsockets was renamed *gsockets*. A significant improvement in both bandwidth and latency was observed using gsockets compared to standard TCP when using NpTest. See Figure 4.

In practice it was found to be very challenging to get a user–space based sockets implementation to work within even the relatively restricted Cray myHadoop usage model. Nonetheless, with sufficient modifications to the various Hadoop startup scripts, and fixes to the original OFED rsockets implementation, it could be made to work with Hadoop. Although no significant speedups of typical Mapreduce jobs has yet been observed using gsockets, it is anticipated that with further testing certain MapReduce jobs may benefit from the use of gsockets. Efforts are underway to rework gsockets into a kernel–based implementation. This should significantly reduce

effort needed to get TCP applications using this higher–performing socket path.

5 Conclusions

The Cray Framework for Hadoop on Cray XC30 provides a supported means by which users can run Hadoop MapReduce applications on Cray XC30 systems in a way that is batch–scheduler friendly. Possible future work to extend the capabilities of the Cray myHadoop include interoperability with Spark [11], enhance the Framework to work with Slurm, as well as enhancements to better work with Lustre at scale.

Acknowledgments

The authors would like to thank James Shimek (Cray, Inc.) and Zach Tiffany (Cray, Inc.) for their work developing the gsockets package.

References

[1] Accumulo. accumulo.apache.org.
[2] Hadoop. hadoop.apache.org.
[3] Myhadoop. sourceforge.net/projects/myhadoop.
[4] *Cray Software Document S-2536-06:Installing myHadoop on Cray XC, XK, and XE Systems*, April 2014.
[5] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar. Puma: Purdue mapreduce benchmarks suite. 2012.
[6] R. Castain, W. Tan, J. Cao, M. Lv, M. Jette, and D. Auble. MapReduce and Support in SLURM: Releasing the Elephant. In *Slurm Users Group 2012*, 2012.
[7] R. H. Castain and O. Kulkarni. MapReduce and Lustre: Running Hadoop in a High Performance Computing Environment. In *Intel Developers Forum 2013*, 2013.
[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[9] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.
[10] M. N. Vora. Hadoop-hbase for large-scale data. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 601–605. IEEE, 2011.
[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. Mccauley, M. Franklin, S. Shenker, and I. Stoica. Fast and interactive analytics over hadoop data with spark. USENIX, 2012.
[12] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, and K. Chen. Survey of mapreduce frame operation in bioinformatics. *Briefings in bioinformatics*, page bbs088, 2013.