Spark on the ARC

Big data analytics frameworks on HPC clusters

Mark E. DeYoung
Virginia Tech
IT Security Lab
Blacksburg, VA 24060, United States
mark.e.deyoung@vt.edu

Mohammed Salman
Virginia Tech
IT Security Lab
Blacksburg, VA 24060, United States
mdsalman@vt.edu

Himanshu Bedi
Virginia Tech
IT Security Lab
Blacksburg, VA 24060, United States
hbedi01@vt.edu

David Raymond
Virginia Tech
IT Security Office & Lab
Blacksburg, VA 24060, United States
raymondd@vt.edu

Joseph G. Tront
Virginia Tech
Bradley Department of Electrical and Computer Engineering
Blacksburg, VA 24060, United States
jgtront@vt.edu

ABSTRACT

In this paper we document our approach to overcoming service discovery and configuration of Apache Hadoop and Spark frameworks with dynamic resource allocations in a batch oriented Advanced Research Computing (ARC) High Performance Computing (HPC) environment. ARC efforts have produced a wide variety of HPC architectures. A common HPC architectural pattern is multi-node compute clusters with low-latency, high-performance interconnect fabrics and shared central storage. This pattern enables processing of workloads with high data co-dependency, frequently solved with message passing interface (MPI) programming models, and then executed as batch jobs. Unfortunately, many HPC programming paradigms are not well suited to big data workloads which are often easily separable. Our approach lowers barriers of entry to HPC environments by enabling end users to utilize Apache Hadoop and Spark frameworks that support big data oriented programming paradigms appropriate for separable workloads in batch oriented HPC environments.

CCS CONCEPTS

•**Information systems** → *Computing platforms;* Nearest-neighbor search; •**Software and its engineering** → *Software as a service orchestration system;*

GENERAL TERMS

Applied Computing, Distributed Computing

KEYWORDS

Advanced Research Computing, High Performance Computing, Big Data

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

PEARC17, New Orleans, LA, USA
© 2017 Copyright held by the owner/author(s). 978-1-4503-5272-7/17/07…$15.00
DOI: 10.1145/3093338.3093375

ACM Reference format:
Mark E. DeYoung, Mohammed Salman, Himanshu Bedi, David Raymond, and Joseph G. Tront. 2017. Spark on the ARC. In *Proceedings of PEARC17, New Orleans, LA, USA, July 09-13, 2017,* 6 pages.
DOI: 10.1145/3093338.3093375

M. DeYoung et al.

1 INTRODUCTION

Developing applications that use existing HPC environments for big data tasks presents several challenges. The programming paradigms for big data frameworks and HPC environments are incongruent and do not precisely overlap. Application level development frameworks for traditional HPC applications are focused on problems with high data co-dependency that can require higher levels of programmer effort[2]. Additionally, there are impedance mismatches between HPC architectures and big data tooling drawn from the Hadoop ecosystem[8]. These mismatches in design paradigms and architectures can be empirically revealed. For example, Spark scaling on HPC systems is examined in [4] which indicates a scalability limit of $O(10^2)$ cores. Regardless, we seek to evaluate the use of HPC clusters with tooling that offers programming models that better align with separable workloads.

In this paper we develop and evaluate deployment models of the Apache Hadoop and Spark frameworks on existing batch oriented HPC clusters. We created a framework to automate the creation of deployment variations and monitor the execution of evaluation iterations that accommodates dynamic resource allocations. We selected the Apache Hadoop and Spark frameworks because they provide programming paradigms that are aligned with the analytical requirements of big data oriented problems where the workload is generally separable (or even arbitrarily separable).

The contributions of our work are:

(1) A framework that enables end-user, on demand provisioning of Apache Hadoop and Spark as programming frameworks in a HPC batch environment.
(2) An evaluation of Apache Hadoop and Spark against standard benchmarks when deployed in an HPC batch environment as a job.

We present an overview of Virginia Tech's (VT) Advance Research Computing (ARC) cluster operations in section 2. Next, we describe our implementation in section 3. In section 4, we evaluate the performance of Apache Hadoop and Spark overlaid on dynamically allocated compute resources by adjusting the deployment to handle service location and service discovery as well as resource and topology discovery. Then in section 6 we discuss two related works. Finally, we provide conclusions and propose future work in section 7.

2 BACKGROUND

Centrally managed HPC clusters at Virginia Tech typically run batch jobs in Portable Batch System (PBS) executed by a Moab workload manager with resources allocation controlled by a TORQUE resource manager[1]. The managed HPC clusters come with a variety of software modules (e.g. compilers, MPI stacks, and high level software) that are user selectable via the Lmod[14] environment modules system[2]. Users submit batch jobs from login nodes command line environment, typically by logging into the login node via a secure shell client.

2.1 Batch Job Submission

A user submits a batch script (in PBS format) which specifies, at minimum, the number of nodes and cores required, the expected wall-clock time the job will run, and a job name. Each cluster offers separate job queues that target specific compute-node types. Also, each queue specifies job admission criteria. After a job is accepted to a queue, Moab advances the job through the job queue until it reaches the top. Then TORQUE allocates the requested resources (at VT using a MINRESOURCE policy). The batch script executes on a head node (AKA "Mother-Superior" node). The batch script can start processes on the head node or via remote access to the worker nodes (AKA "Sister" nodes).

As an example, a batch job with three compute-nodes allocated is shown in Figure 1 and Figure 2 . Figure 1 shows the Hadoop NameNode (NN) and YARN ResourceManager (RM) service daemons running on the head node. Each of the service daemons is executed in a Java Virtual Machine (JVM). Each JVM is contained by a Unix process. Figure 2 shows the Hadoop DataNode (DN) and YARN NodeManager (NM) running on each of the worker compute-nodes allocated to the job. Like the daemons on the mother-superior, the Hadoop/YARN daemons are executed by JVMs contained within Unix processes.

The PBS job describes the resource requirements a user expects will result in a runtime environment with sufficient resources capable of running the user's application. Dynamically adjustable or configurable runtime environments (i.e. virtual machines, containers, and application level virtual machines like the JVM) present deployment challenges. The application configuration must be adjusted to leverage available resources. In a static deployment model resources are known a priori so the application configuration is created before the application is executed.

[1]TORQUE and Moab details can be found in [5] and [6] respectively. Additional, documentation is at [7].
[2]VT ARC Software Modules - https://secure.hosting.vt.edu/www.arc.vt.edu/software-modules/

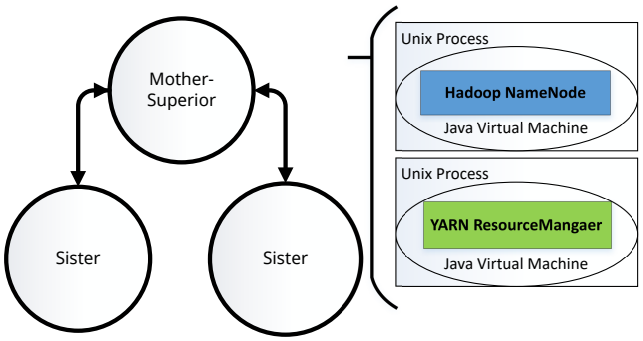

Figure 1: Mother-Superior (head node) running Hadoop NameNode and YARN ResourceManager daemons.

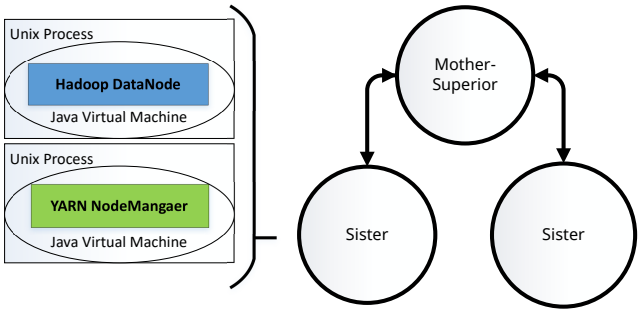

Figure 2: Sister (worker node) running Hadoop DataNode and YARN NodeManager daemons.

2.2 Programming Paradigms

While HPC clusters are typically designed for workloads with high data co-dependency, often programmed with MPI approaches, our intuition is that provisioning the Spark framework will lower barriers of entry for users with big data type separable workloads. We seek to overcome an impedance mismatch between workload and framework while leveraging existing compute resources. The Hadoop ecosystem contains a variety of big data oriented frameworks and tooling. We selected the Apache Hadoop and Spark frameworks because they are core applications that also support purpose focused extensions for machine learning, text mining, graph analysis, and data manipulation (i.e. ability to extract, transform, and load unstructured data sources)[16].

Because we are deploying the frameworks without elevated permissions we must dynamically generate a PBS script and configuration with competing goals:

- Be a good citizen
- Achieve acceptable performance

Be a good citizen - don't abuse shared resources by over subscribing from the shared HPC cluster. That is, don't create jobs that request excessive resources for the expected workload or in a manner that denies other users access to shared resources. At the same time we desire that our jobs complete in a timely manner.

An Apache Hadoop or Spark cluster deployment is comprised of several interacting services that communicate and coordinate action over network connection which are typically TCP/IP based[3]. Some of the core services for Hadoop/YARN are shown in figure 1 and figure 2.

Two common distributed systems design issues were quickly revealed. The framework services must be able to locate each other, so they can establish communication for coordination of work effort; and resource allocations are not static. We need means to discover services and resource topology so we can encode the configuration parameters for service daemons and then inject resource aware configuration into the allocated runtime environment.

2.3 Service Discovery and Service Location

More formally, we must solve a *Service Location and Service Discovery* problem for a distributed system overlaid onto a set of resources. There are several ways to approach service discovery and service location: static placement during provisioning with service locations encoded in configuration files (e.g. Hadoop is often provisioned onto a dedicated set of compute-nodes with known network configurations); dynamic discovery with registration in a service directory (e.g. mini-DNS); or hybrids[4]. The TORQUE resource manager provides an environment variable $PBS_NODES_FILE that contains the location of a file that can be parsed into a list of host names for the compute-nodes allocated to a specific job. The parsing technique is used in Open MPI and myHadoop[13].

2.4 Resource and Topology Discovery

Hardware topology is known to impact distributed systems performance. Apache Hadoop and Spark can both be configured to leverage network topology to reduce communications overhead. We must also adjust some aspects of the deployment to accommodate resource topology and resource availability, that is we must also address *Resource and Topology Discovery*. While the TORQUE/Moab scheduling process can be configured to accommodate some locality in job descriptions the configuration at VT does not incorporate this into scheduling decisions. Also, the TORQUE resource manager and Moab workload manager do not handle availability of some resources (e.g. TCP/IP port allocations) which means this must be handled by the frameworks daemons when they begin execution on allocated compute-nodes.

3 IMPLEMENTATION

Our implementation and design strategy was driven by the types of workloads we envision are appropriate for the system under design. Additionally, our design goals include: ease of use for novice HPC users, no use of elevated privileges, and non-persistent deployment. We intend to improve accessibility to HPC cluster resources by enabling the use of alternative programming paradigms that lower barriers of entry.

Traditional HPC clusters are architected around shared storage like the open source Lustre File System [5] or IBM's General Parallel File System (GPFS)[6] with high performance interconnect fabric such as, InfiniBand or Intel Omni-Path Architecture (OPA).

At the other extreme, Hadoop and Spark are designed for 'shared nothing architecture'. Workloads are generally: uniform - composed of identically sized segments; modular - segmented a priori according to some extrinsic knowledge about the data; arbitrary - workload can be segmented arbitrarily. This leads to some impedance mismatches between the Hadoop frameworks and HPC compute architecture. Additionally, Hadoop uses the Apache Yet Another Resource Negotiator (YARN) to manage resources across the running framework cluster. This means our approach will incur overhead costs of an additional resource management framework within the context of the PBS job runtime environment.

3.1 Software and Data requirements

We implemented our framework to support Apache Hadoop version 2.7.3 (which includes YARN), and Spark version 2.1.0. Virginia Tech's ARC HPC clusters use Adaptive Computing's TORQUE resource manager version 4.2 and Moab workload manager version 7.2 for resource management and scheduling. We generated data for our analysis with Hadoop Bench and Spark Bench drawn from the HiBench big data benchmark suite[9].

3.2 Design Strategy

Enabling additional programming models and dynamic deployment offers the opportunity to improve resource utilization of HPC clusters. Dynamic deployment sizes support scaling the resource requirement to the expected workload. Centrally administered environments, like VT's ARC HPC clusters, are designed around many tradeoffs. One of the tradeoffs we considered in our design is user desire for flexibility (different versions of software and tools) vs. IT controls for multi-tenancy concerns (e.g. data security and network isolation). Our approach to this tradeoff is to deploy our framework in a manner that does not require elevated privileges and is not persistent in the runtime environment. That is, we do not permanently install software but instead deploy the software as needed and configure it to the runtime environment resulting from the resources allocated to the batch job.

VT ARC clusters are a shared resource where multiple jobs can be scheduled on each compute-node. Because of this we should consider the impact of multi-tenant contention for resources. Additionally, the clusters are configured with limits for specific resources. The resource limits are enforced before a job is admitted into a queue. Furthermore, because the environment is configured with a MINRESOURCE policy we must create job descriptions that minimize resource utilization so they are likely to be scheduled to run in an acceptable time frame. Our design addressed the following limiting factors:

- no elevated privileges
- fixed batch scheduling policies
- limited locality awareness
- no internet access from compute-nodes

To dynamically instantiate either Spark or Hadoop in a cluster mode on ARC, we have written a framework along the lines of the

[3]Portions of the Apache Hadoop stack, including Spark, were modified to use Remote Direct Memory Access (RDMA) instead of TCP/IP by [15] and [10].

[4]As and example, the Apache Ambari project at https://ambari.apache.org/ supports dynamic provisioning of clusters onto a static set of compute-nodes.

[5]Lustre - http://lustre.org/about

[6]GPFS is now named IBM Spectrum Scale. See: http://www-03.ibm.com/systems/storage/spectrum/scale/

myHadoop framework[12, 13]. Both Hadoop and Spark require Java to be installed or the Java module to be loaded. The basic PBS job workflow, similar to that used in myHadoop, is:

(1) Discover compute-nodes allocated
(2) Configure the Mother-Superior node as head and the Sister nodes as workers.
(3) Format Hadoop NameNode (NN)
(4) Start Hadoop services: NameNode on Mother Superior, DataNode on Sisters
(5) Start YARN services: ResourceManager on Mother Superior, NodeManager on Sisters
(6) Stage-in Data
(7) Load the required modules for the application (e.g. Java environment, Python with numpy, etc.)
(8) Run User Application (in our case HiBench[7])
(9) Stage-out Results
(10) Terminate after shutting down Hadoop and YARN services

Unlike myHadoop, we are only considering deployment with storage local to each of the compute-node. The Hadoop File System (HDFS) will be distributed across Sister (worker) nodes. This means the Hadoop DataNode daemons are using local storage on each of the Sister (worker) nodes. Modifications to the HDFS are non-persistent, that is, they will not be available after the PBS job completes because we will no longer have access to the compute-nodes local storage. Likewise the YARN NodeManager daemons execute tasks on Sister (worker) nodes as shown in figure 1 and figure 2 in section 2. Tasks scheduled to execute will be terminated at the end of the PBS job resulting in loss of any incomplete computation.

In order to configure a standalone Spark cluster on a distributed file system like GPFS, the PBS job workflow is as follows:

(1) Discover compute-nodes allocated
(2) Copy the Spark software to a shared folder which is visible from all nodes. In case there is no distributed file system, either set up HDFS or manually copy Spark to each of the Sister nodes.
(3) Select one of the nodes as master(in our case the Mother-Superior); run the start_master.sh script; and export an environment variable containing the Uniform Resource Locator (URL) for the Spark cluster master.
(4) Configure the Sister nodes as workers by starting the start_slaves.sh script.
(5) In order to get maximum performance, the Spark parameters can be varied. Currently, we rely on the Spark scheduler to distribute jobs efficiently. However, as part of future work we intend to build a resource request manager which profiles the jobs run by the user and computes the resources required efficiently.
(6) The telemetry results for the job performance(CPU consumption, disk utilization, job queuing delay) is collected using programs included with the TORQUE/Moab installation.

Our framework implementation targeted four of Virginia Tech's six centrally managed ARC HPC clusters: BlueRidge, Cascades, DragonsTooth, NewRiver. Details of the cluster configurations are available from VT's ARC online documentation[8]. In summary: BlueRidge is a Cray CS-300 with 408 Intel Xeon E5-2670 (Sandy Bridge) compute-nodes designed for large-scale compute intensive jobs; Cascades is oriented toward data intensive problems with 196 Intel Xeon E5-2683v4 (Broadwell) compute-nodes; DragonsTooth has 48 Intel Xeon E5-2680v3 (Haswell) compute-nodes and is intended to support long-running single node jobs; finally, NewRiver has 134 Intel E5-2680v3 nodes is also oriented toward data intensive problems. Each of the clusters also has specialized nodes that offer advanced features. For example Cascades has two compute-nodes with additional memory (3 TB) and four compute-nodes with NVIDIA K80 GPUs. We did not leverage any of the special features of specific compute-nodes.

4 EVALUATION

For our evaluation we conducted multiple iterations where we measured and characterized performance of a variety of configuration models to demonstrate feasibility of our approach. While we validated our framework functionality on the four clusters previously mentioned we did not examine the performance in detail on all four. Evaluation was carried out on two clusters maintained by VT's ARC, namely Cascades and NewRiver, whose characteristics are highlighted in table 1 and table 2 respectively. As mentioned in 3, jobs are initiated on compute-nodes by user submitted Portable Batch System (PBS) scripts.

Table 1: VT ARC cluster Cascades

Compute Engine	No of Nodes	Type of CPU	Cores	Memory
General	190	2 x E5-2683v4 2.1GHz(Broadwell)	32	128GB
GPU	8	2 x E5-2683v4 2.1GHz(Broadwell)	32	512 GB
Very Large Memory	2	4 x E7-8867v4 2.4GHz(Broadwell)	72	3 TB

Table 2: VT ARC cluster New River

Compute Engine	No of Nodes	Type of CPU	Cores	Memory
General	100	2 x E5-2680v3 2.5GHz(Haswell)	24	128GB
Big Data	16	2 x E5-2680v3 2.5GHz(Haswell)	24	512 GB
GPU	8	2 x E5-2680v3 2.5GHz(Haswell)	24	512 GB
Interactive	8	2 x E5-2680v3 2.5GHz(Haswell)	24	256 GB
Very Large Memory	2	4 x E7-4890v2 2.8GHz(Ivy Bridge)	60	3 TB

A dynamic Spark and Hadoop cluster is instantiated and the scheduling is carried out in both the standalone mode and with YARN. We ran two benchmarks- namely Spark Bench and Hadoop Bench to test the Spark and Hadoop configurations. Leveraging the telemetry framework that VT ARC provides as part of the TORQUE/Moab installation, we collected telemetry data which includes: the queuing delay, time to completion, CPU utilization and memory consumption. Further, the analysis is done by varying the HPC resources again using the framework that VT ARC provides. We investigate the effects of horizontal scaling versus vertical scaling by comparing the resource utilization in either case.

[7] HiBench[9]

[8] VT ARC Compute - https://secure.hosting.vt.edu/www.arc.vt.edu/computing/

Horizontal scaling (scale out) implies adding more number of nodes in the cluster to run the jobs while vertical scaling (scale up) implies requesting more cores on the same node to run the jobs.

Using Spark Bench, we ran three benchmarks- kmeans, pagerank and logistic regression while in Hadoop Bench, we ran the entire gamut of benchmarks provided. The benchmarks were selected to evaluate the overhead of inter-node communication in compute intensive jobs and I/O intensive jobs.

The telemetry data is collected using a script which collects the CPU consumption, memory consumption,queuing delay as the job executes and the avg consumption parameters once the job finishes running. The Pseudo-code for the telemetry script is as follows-

```
jobload $job_id
result="start"
while [ $job_id is running ]
do
    sleep t seconds
    result=`jobload $job_id`
    jobload $job_id >> $job_id.txt
done
checkjob -v $job_id >> $job_id.txt
```

5 EXPERIMENT RESULTS

In case of Spark Bench, the tests were run on 100 million data points while in case of Hadoop Bench, we used the huge data profile provided by the Hibench framework. Figure 3 shows the Hadoop Bench results for different jobs when the number of resources are horizontally scaled out and figure 4 shows the Hadoop Bench results for different jobs when the same node is scaled up vertically in terms of number of nodes. Figure 5 shows the Spark Bench results in case of horizontal scaling and figure 6 shows the Spark Bench results in case of vertical scaling. From the results, over-committing of resources is evident which explains the flat lines in terms of the time to completion for the jobs.

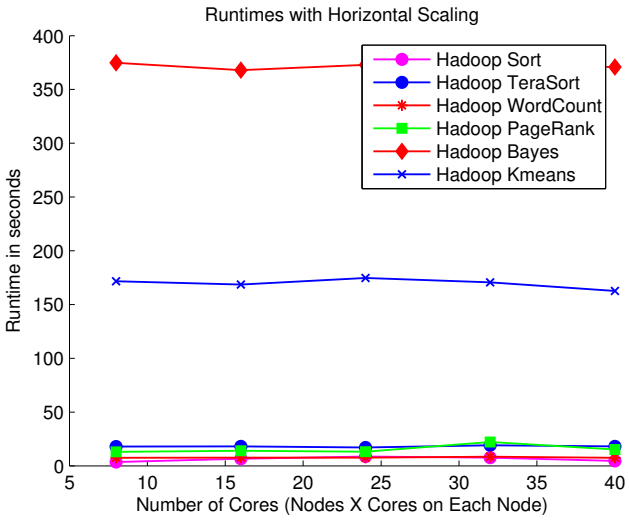

Figure 3: Hadoop Horizontal Scaling

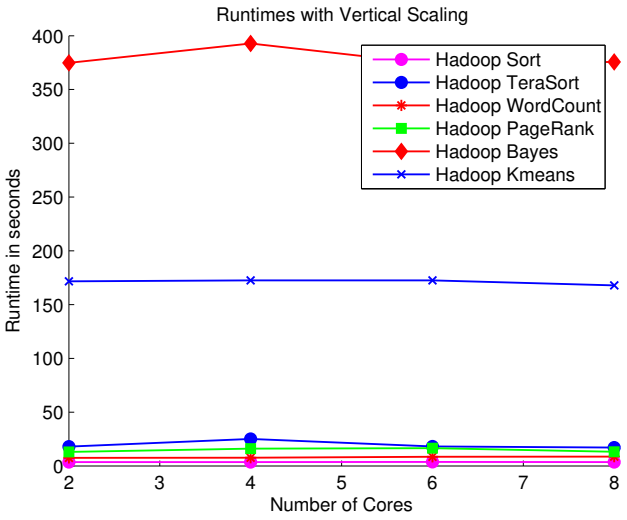

Figure 4: Hadoop Vertical Scaling

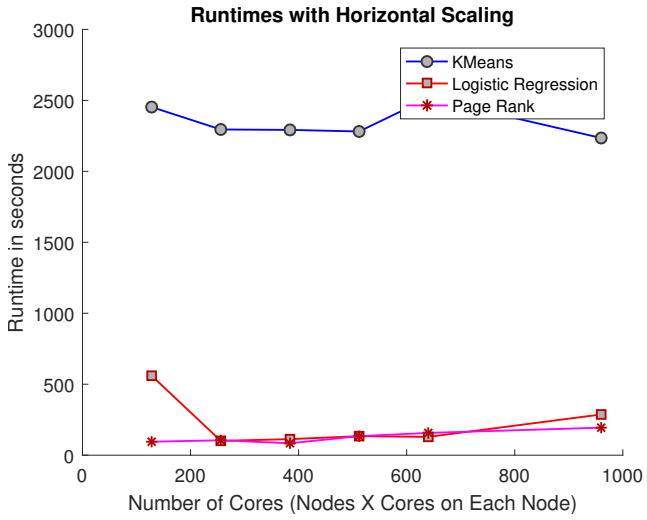

Figure 5: Spark Horizontal Scaling

6 RELATED WORK

Two related works are myHadoop and `pbs-spark-submit`. The myHadoop framework provisions a Hadoop cluster in the resources allocated to a PBS batch job [12, 13]. As previously discussed in section 3.2 we did not implement a persistent data option but might do so in future work. Baer,et. al. integrated Apache Spark into a TORQUE PBS batch system as a Python script (`pbs-spark-submit`)[3]. Additionally, they ran multiple example programs includes with the Spark distribution as micro-benchmark tasks including: Spark Pi, Spark SQL, Spark PageRank, and Spark Wordcount. Similar to our work they did not modify Spark to account for multi-tenancy concerns. Unlike our approach they did not attempt to manage resources (i.e. TCP/IP ports) that are not controlled by TORQUE/Moab.

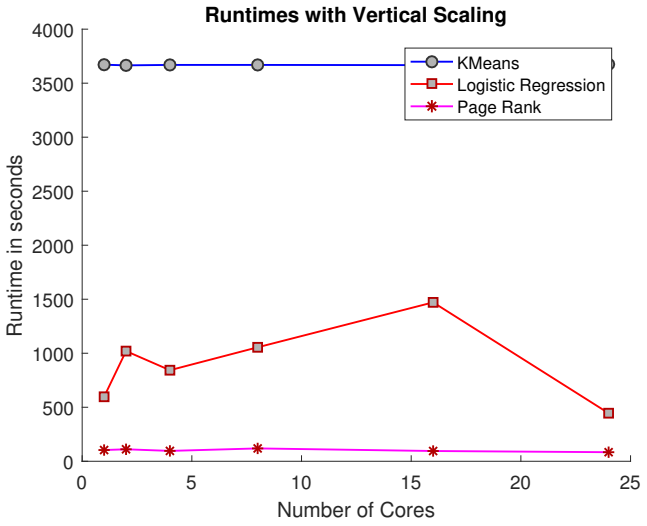

Figure 6: Spark Vertical Scaling

Additionally, their work was focused on Spark and did not include other components like Apache Hadoop and YARN.

7 CONCLUSIONS

MPI focused HPC clusters do not offer a full range of capabilities that enable distributed systems in varied tiered and layered deployment models. The overhead of dynamically commissioning frameworks, deploying applications, and staging data does not preclude their use for some workloads. With that said there are several limitations in our work that should be considered. While we have demonstrated feasibility of deploying Apache Hadoop and Spark in a YARN configuration we did not comprehensively evaluate the overhead associated with the deployment, nor did we evaluate the impact of user contention when the framework is deployed on to compute-nodes that are shared (i.e. they are running processes from multiple users jobs). Furthermore, our scaling results are based on a small number of job runs for generative workloads.

There is much room for future work now that we have established the ability for end-users, like ourselves, to provision Apache Hadoop and Spark as programming frameworks in a HPC batch environment. Some areas to examine include examining overhead incurred, more extensive benchmark evaluation, and more importantly beginning to run realistic, real-world workloads. For example, we intend to use the framework we have created to conduct network security data analytics. We will also consider analysis of streaming data and scheduling of heterogeneous node types (leveraging special features like GPUs, Xeon Phi many integrated core co-processors, and high performance node local storage) like [11]. In addition, an evaluation of energy efficiency based on scheduling various workloads can also be carried out similar to [1].

ACKNOWLEDGMENTS

The authors acknowledge Advanced Research Computing at Virginia Tech for providing computational resources and technical support that have contributed to the results reported within this paper. URL: http://www.arc.vt.edu. Also, the authors would like to thank the Virginia Tech Open Access Subvention Fund (VT OSAF) for enabling publication as an open access document. URL: http://www.lib.vt.edu/oafund

REFERENCES

[1] Ali Anwar, KR Krish, and Ali R Butt. 2014. On the use of microservers in supporting hadoop applications. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 66–74.

[2] H. Asaadi, D. Khaldi, and B. Chapman. 2016. A Comparative Survey of the HPC and Big Data Paradigms: Analysis and Experiments. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 423–432. DOI: http://dx.doi.org/10.1109/CLUSTER.2016.21

[3] Troy Baer, Paul Peltz, Junqi Yin, and Edmon Begoli. 2015. Integrating Apache Spark into PBS-Based HPC Environments. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure (XSEDE '15)*. ACM, New York, NY, USA, 34:1–34:7. DOI: http://dx.doi.org/10.1145/2792745.2792779

[4] Nicholas Chaimov, Allen Malony, Shane Canon, Costin Iancu, Khaled Z. Ibrahim, and Jay Srinivasan. 2016. Scaling Spark on HPC Systems. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '16)*. ACM, New York, NY, USA, 97–110. DOI: http://dx.doi.org/10.1145/2907294.2907310

[5] Adaptive Computing. 2015. Moab Workload Manager Administrators Guide 7.2.10. (March 2015). http://docs.adaptivecomputing.com/mwm/7-2-10/mwmAdminGuide-7.2.10.pdf

[6] Adaptive Computing. 2015. TORQUE Resource Manager Administrator Guide 4.2.10. (March 2015). http://docs.adaptivecomputing.com/torque/4-2-10/torqueAdminGuide-4.2.10.pdf

[7] Adaptive Computing. 2016. Documentation Index. (2016). http://www.adaptivecomputing.com/support/documentation-index/

[8] Nicole Hemsoth. 2015. Just How Deep is the HPC, Hadoop Chasm? (Aug. 2015). https://www.nextplatform.com/2015/08/03/just-how-deep-is-the-hpc-hadoop-chasm/

[9] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. IEEE, 41–51. DOI: http://dx.doi.org/10.1109/ICDEW.2010.5452747

[10] Nusrat S. Islam, M. W. Rahman, Jithin Jose, Raghunath Rajachandrasekar, Hao Wang, Hari Subramoni, Chet Murthy, and Dhabaleswar K. Panda. 2012. High performance RDMA-based design of HDFS over InfiniBand. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 35. http://dl.acm.org/citation.cfm?id=2389044

[11] KR Krish, Ali Anwar, and Ali R Butt. 2014. [phi] sched: A heterogeneity-aware hadoop workflow scheduler. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*. IEEE, 255–264.

[12] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. 2011. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego* (2011).

[13] Glen Lockwood. 2012. glennklockwood/myhadoop. (2012). https://github.com/glennklockwood/myhadoop

[14] Robert McLay. 2011. Lmod: Environmental Modules System. (2011). https://www.tacc.utexas.edu/research-development/tacc-projects/lmod

[15] Dhabaleswar K. Panda. 2016. High-Performance Big Data :: Home. (2016). http://hibd.cse.ohio-state.edu/

[16] Javi Roman. 2014. The Hadoop Ecosystem Table. (Jan. 2014). https://hadoopecosystemtable.github.io/