Building a Research Data Science Platform from Industrial Machines

Fang (Cherry) Liu*, Fu Shen‡, Duen Horng Chau‡, Neil Bright* and Mehmet Belgin*
*Partnership for An Advanced Computing Environment (PACE)
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
Email: fang.liu@gatech.edu
‡School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
Email: polo@gatech.edu

***Abstract*—Data Science research has a long history in academia which spans from large-scale data management, to data mining and data analysis using technologies from database management systems (DBMS's). While traditional HPC offers tools on leveraging existing technologies with data processing needs, the large volume of data and the speed of data generation pose significant challenges. Using the Hadoop platform and tools built on top of it drew immense interest from academia after it gained success in industry.**

Georgia Institute of Technology received a donation of 200 compute nodes from Yahoo. Turning these industrial machines into a research Data Science Platform (DSP) poses unique challenges, such as: nontrivial hardware design decisions, configuration tool choices, node integration into existing HPC infrastructure, partitioning resource to meet different application needs, software stack choices, etc.

We have 40 nodes up and running, 24 running as a Hadoop and Spark cluster, 12 running as a HBase and OpenTSDB cluster, the others running as service nodes. We successfully tested it against Spark Machine Learning algorithms using a 88GB image dataset, Spark DataFrame and GraphFrame with a Wikipedia dataset, and Hadoop MapReduce wordcount on a 300GB dataset. The OpenTSDB cluster is for real-time time series data ingestion and storage for sensor data. We are working on bringing up more nodes. We share our first-hand experience gained in our journey, which we believe will benefit and inspire other academic institutions.

***Keywords*-Data Science Platform; Hadoop; Spark; Software and Hardware Configuration; Machine Learning**

I. Introduction

The Hadoop software library provides a framework to process large data sets distributed across clusters of computers [22]. It uses simple programming modules and hides the complexity of inter-computer communication and data transformation. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The Hadoop library is designed to detect and handle failures at the application layer, so that commodity hardware can be used without worrying about node downtime. The framework has several modules:

- Hadoop Distributed File System (HDFS): a distributed file system for high throughput data access.
- Hadoop YARN: job scheduling framework and cluster resource management.
- Hadoop MapReduce: YARN-based system for parallel processing of large data sets.
- Hadoop Common: utilities supporting other modules.

There are various ways to use Hadoop without an in-house cluster. Amazon Elastic Compute Cloud (Amazon EC2) [2] is a web service that provides resizable compute capacity in the cloud. The advantage to use Amazon EC2 is that user has complete control of EC2 instances with given root access, and the user can retain the data on root partition when stopping and restarting the instance. All operations can be done using web service APIs. The user also has access to the console output of EC2 instances. However system admin knowledge is required for Hadoop to run efficiently.

Amazon Elastic MapReduce (Amazon EMR) [3] simplifies big data processing with a managed Hadoop framework. It supports popular tools in the Hadoop ecosystem, such as Hive [6], Pig [7], HBase [5], and Impala [11]. Additionally, it can run distributed computing frameworks besides Hadoop MapReduce, such as Spark or Presto. Hue and Zeppelin have be used as GUIs for interacting with applications on the cluster. EMR is a collection of EC2 instances with Hadoop (and optionally other tools) installed and configured on them. Amazon EMR costs less to use [19] than Amazon EC2. However, using EMR does not benefit from the Hadoop Distributed File System (HDFS) since it involves storing data on Amazon's S3 which introduces a higher latency and slower file I/O. Using HDFS on EMR requires leaving the cluster running to preserve the data, which raises costs [1].

Microsoft Azure Data Lake Store (DLS) is an Apache Hadoop file system compatible with HDFS and works with the Apache Hadoop ecosystem. It is integrated with Azure Data Lake Analytics (DLA) and HDInsight [17] that provides a managed Apache Hadoop, Spark, R, HBase and Storm cloud service with language support including C#, Java and .NET. However, Azure uses Hortonworks Data Platform (HDP), which often does not include the most recent software releases from the Hadoop ecosystem [15]. Based our own experience, HDInsight instance costs higher than Amazon EMR.

Google Cloud Platform (GCP) allows a general developer to take advantage of rich machine learning capabilities with simple representational state transfer (REST) based services. Google Cloud Dataproc (GCD) provides an Apache Hadoop, Spark, Pig and Hive service, and it is also integrated across GCP products. Comparing to Amazon and Microsoft, GCP provides higher IOPS as storage using HDD and SSD [20], but offer fewer types of machine instances.

Multiple courses at Georgia Tech have been using Amazon EMR, Amazon EC2 and HDInsight in the classroom for assignments and projects. However, in order to fully understand the Hadoop ecosystem and modify it for research, public cloud services might not meet the requirements due to delayed access to up-to-date or custom software, potential high costs of long running services, possible limitations on performance (e.g., Amazon S3), etc. Building an Apache Hadoop ecosystem in-house, using existing or donated nodes (e.g., from yahoo), becomes an attractive solution.

In this paper, we will first discuss the overall goals and considerations in building the DSP. Next, we describe the hardware decisions and software choices that we made. Then, we present experiment results to demonstrate the usage of DSP. Lastly, we conclude. A short abstract of this work has been accepted by the Women in HPC SC16 workshop [23]. The first author will present a poster there.

II. Challenges to Address

To achieve our goal for a successful cluster deployment, we identified several core needs by surveying the requirements of target research applications that DSP aims to support:

- Performance: how to get most performance out of existing hardware.
- Maintenance: how to make maintenance as less intrusive as possible.
- Sustainability: how to enable the whole system scale up to more compute nodes in the future.

Concretely, a few research projects at Georgia Tech are using Spark [8] for streaming data processing, and using HBase [5] and OpenTSDB [18] to store time series sensor data from power generating assets. As these projects tend to require large memory and random data accesses, memory size per node and disk I/O may have higher priority than the amount of total storage. On the software side, there are also multiple important considerations, such as: How to allow the new DSP co-live with existing HPC system? How to embody Hadoop configuration in the current system's configuration stream? Should we choose tightly-coupled schemes or loosely-coupled schemes? How to make future maintenance less error-prone? When considering the initial hardware setup, we need to take into account sustainable future growth for the system, such as hard drive sizes of the service nodes, etc.

III. Hardware and Architecture

In this section, we will describe our preliminary system architecture and its hardware. We received a donation of 200 compute nodes from Yahoo. Each node is equipped with dual 4-core Intel Xeon CPUs (i.e., 8 cores total) at 2.50GHz, 24GB memory, and two 500GB hard drives. We are using the existing hard drives for slave nodes. We upgraded the service nodes to equip each one with two 1TB hard drives. In order to more efficiently utilize the given hardware to build a durable Hadoop cluster, we need to answer several setup questions:

- Should a NameNode (NN) be configured the same way as a DataNode (DN) in hardware?
- What is the best disk partitioning scheme for a DN and a NN?
- Where to put *hadoop home*.
- Where to put Datanode's *data* directory.
- Where to put Namenode's *name* directory.
- Where to put the *hadoop log* directory.

A. Hadoop System Requirement

Hadoop distributes data across a cluster of balanced machines and uses replication to ensure data reliability and fault tolerance. Because data is distributed on machines with compute power, processing can be performed directly on those machines. Since each machine in a Hadoop cluster simultaneously stores and processes data, it needs to be configured to satisfy both data storage and processing requirements.

The Hadoop Distributed File System (HDFS) is platform independent and can function on top of any underlying file system and Operating System (OS). There are two types of file system, Ext3 and Ext4, from which we need to choose one. Ext3 lacks a lot of features that Ext4 provides, the most significant limitation is that it cannot support file size larger than 2TB. However, this does not significantly impact HDFS since it is unlikely for the block size to be set to 2TB. Hadoop has a default setting on the minimum block size in bytes (1MB), enforced by the NameNode at creation time. This prevents accidental creation of files with tiny block sizes, which degrade performance. The maximum number of blocks per file (1 million blocks) is enforced by the NameNode on write time of files which prevents the creation of extremely large files. With 1M blocks of 1MB file, Hadoop can easily handle file size of 1TB. Also, as Ext3 has been used by Yahoo for its internal Hadoop deployment with proven stability, it makes Ext3 a safe choice for our underlying file system.

DataNodes usually boot from a small, dedicated partition on /dev/sda. However, logs and caches can rapidly fill a partition. In general, people choose to use separate disks for the OS and data disks used by DataNode (DN) and ResourceManager (RM) to avoid I/O contention as well as the imbalance in space consumption. At least one data disk is needed, and multiple data disks are preferred. Hadoop tends to generate a lot of log files. Putting the logs on the OS disk instead of the data disks will avoid performance degradation of the drive, in hence, lead to slower computation. In most cases, one needs not worry about OS disk failures since Hadoop is designed to handle individual node failure. If the OS partition on a node fills up, the whole cluster will still keep running.

The most common types of computer storage are redundant array of independent disks (RAID) 0 (striping), RAID 1 (mirroring), RAID 5 (distributed parity), and RAID 6 (dual parity). RAID 0 splits data evenly across two or more disks, without parity information, redundancy, or fault tolerance. RAID 1 creates an exact copy (or mirror) of a set of data on two or more disks, and the service will continue to operate as long as at least one member drive is operational. For service nodes such as NameNode (NN), Secondary NameNode (NN) and ResourceManager (RM), RAID 5/6 is generally recommended. However the hardware available in our hands only has two slots, which does not meet the minimal requirement of RAID 5/6, so RAID 1 was chosen for service nodes. And RAID disk controller for DN is not needed, as Hadoop replicates data across multiple machines. It is also a good idea to put DNs on different PSUs and switches to further reduce points of failure in the data center. While the Hadoop NN and Secondary NN can write to a list of drive locations, they will stop functioning if none of the locations can be written to. Thus, a mirrored RAID is a good idea for higher disk availability.

The NN is responsible for coordinating data storage on the cluster, and the RM for coordinating data processing. The Secondary NN should use hardware identical to that of the NN, but should not co-locate with the NN.

B. Hardware configuration

Based on the above requirement analysis, we configure our hardware to maximize performance. As a node's computation speed greatly depends on the machine's available memory, we filled each node with 24GB RAM, the maximum amount that it can support. Depending on how many hard drives the system has, the Operating System (OS) drive should be separated from storage/data drive to reduce I/O contention. We chose RAID 1 for service node on a mirrored 1TB hard drive and chose to not use RAID for slave nodes with one 500GB hard drive dedicated to the OS while another 500GB hard drive dedicated to data storage, mounted as $/dfs$. The Hadoop home is put under $/dfs/hadoop$, DataNode's *data* directory is set under $/dfs/hadoop/data$ while NameNode's *name* directory is created under $/dfs/hadoop/name$. Hadoop is known to generate a lot of log files, so we enlarge the $/var$ directory to $100TB$ to accommodate the log files.

Two clusters have been built so far:

- A 24-node Hadoop and Spark cluster (HSC) is deployed with one NN, one Secondary NN sitting on the same service node with RM and 22 DNs. Spark has been configured to run on top of YARN, so that researchers can test out both MapReduce jobs and Spark jobs on the same cluster.
- A 12-node HBase and OpenTSDB cluster (HOC) is established with one NN, one Secondary NN for HDFS,. HBase has been configured with one HMaster, and one Backup HMaster and 10 regionservers, and HBase uses the build-in Apache zookeepers [9] instead of third party zookeepers since the cluster is small.

In order to avoid heating issues, a decision has been made to separate the two clusters into two racks, as the optimal nodes per rack should be around 30.

IV. Software and Configuration

One of the first tasks when planning a Hadoop deployment is to select the distribution and version of Hadoop that is most appropriate, given the features and stability required. This process requires input from users who will eventually use the cluster: system administrators, developers and researchers. Hadoop is, as previously mentioned, an Apache Software Foundation (ASF) project, in which its source and binary are freely available from Apache. Although Hadoop provides both distributed file-system and MapReduce processing framework, many users view it as the core of a larger system, which is analogous to an operating system kernel, upon which higher-level systems and tools can be built.

A. Hortonworks, Cloudera or Apache

Hadoop is an open source project and several vendors have been developing their own distributions on top of the Hadoop framework to make it enterprise-ready, supplementing it with enterprise reliability and integration and management capabilities. While these custom extensions provide additional functionality, they often reduce flexibility on the user's side.

Hortonworks Data Platform (HDP) [14], founded by Yahoo engineers, provides a "service only" distribution model for Hadoop. It is the only Hadoop Distribution that supports the Windows platform. Its Ambari management interface replaces Hue [16] for supporting both cluster monitoring and data analysis. However, Hortonworks does not commit the code change back to Apache Hadoop to avoid vendor lock-in. HortonWorks' Implementation, Advisory, and Managed (IAM) services are provided to help users adopt HDP.

Cloudera [12], a company that provides support, consulting, and management tools for Hadoop, also has a distribution of software called Clouderas Distribution Including Apache Hadoop (CDH) which is available under the Apache Software License and is free for both personal and commercial use and it backports critical fixes to Apache Hadoop release as well. CDH includes most major components from the Apache Hadoop ecosystem, and also provides useful tools such as Cloudera Impala [11]. Its management console – Cloudera Manager, implemented with a rich user interface, is easy to use. The Cloudera Management suite automates the installation process and also renders various other enhanced services to users to reduce deployment time. Cloudera offers consulting services to bridge the gap between what the community provides and what organizations need to integrate Hadoop technology into their data management strategy.

The drawbacks of enterprise-ready Hadoop distributions:

- Automated installation is hard to blend into exiting software management scheme within HPC systems (such as those at Georgia Tech). In order to manage software in a large system, usually a configuration tool is used to specify installation locations, which allows future software update, removal, etc. However, HortonWorks's Ambari requires all software to be installed in the system's default directories, which makes it very hard to clean the system and keep track software locations. HDP allows users to alter the installation directory, but the user has to prepare a long list of environment variables to control those locations.
- Infrequent package release cycles delay or prevent the usage of up-to-date software (e.g., HortonWorks only releases twice a year). Researchers who want to experiment with the latest software features may be hesitant to adopt it.
- Not all Apache software libraries are available in enterprise release, it limits the researcher to try new tools.
- Consulting service is fee-based, and the tools' dependency is built into the packages which makes it harder to debug when there are any issues, and makes users more dependent on the consulting service. This makes non-funded project hard to survive.

It is common to adopt open source software than commercial software in academia due to the freedom to use up-to-date source code, to tailor the code to meet local requirements, to reduce costs that may be hard to budget for, to configure the system to work with existing systems, etc. We decided to use Apache Hadoop 2.7.2 for ongoing research based on above reasoning.

For other tools, we used their latest versions at the time: HBase 1.1.5, OpenTSDB 2.2.0 and Spark 1.6.1.

B. *Tuning Hadoop*

To enable a Hadoop cluster to run in optimal ways, a lot of tuning is needed. YARN – the resource manager takes into account all of the available compute resources on each machine in the cluster, and negotiates resource requests from applications running in the cluster such as MapReduce. It then provides processing capacity to each application by allocating Containers, which are the basic units of processing capacity. A Container encapsulates resources elements such as memory and CPU. Each of our machines has 24GB RAM. Based on the reserved memory recommendations of HortonWorks HDP, 4GB should be reserved for system memory, and another 4GB should be reserved for HBase, which leaves 16GB for RM. However, separating MapReduce and HBase may reduce some memory contention. The number of files, the number of blocks and the load on the system may affect NameNode's heap size choice. For our research projects, we expect the number of files to be fewer than 1 million, so Java Heap (Xmx and Xms) can be set to $1028m$. This setting hasn't been fully tested yet, further tuning may be needed if we encounter performance degradation during run time.

V. System Configuration

Data center computer system configuration can be performed using a collection of scripts or via some automation frameworks. The former method may be harder to manage for large number of computer nodes; the latter option may involve a steep learning curve. For our clusters, we are managing groups of identical servers, running identical applications and services. What we need is a consistent, reliable, and secure way to manage the environment. Our system uses a loosely-coupled approach to preserve both security and freedom of management, using Puppet [21] to configure the machines' OS, and to provision the bare metal. And it uses another tool (which we shall discuss next) to configure Hadoop related settings.

A. *Ansible, Puppet or Chef*

Publicly available tools such as Ansible [4], Puppet, and Chef[10] are widely used for system configuration. Ansible is the simplest solution for configuration management available. It can be run from the command line without configuration files for simple tasks, such as making sure a service is running, or to trigger updates and reboots. For more complex tasks, Ansible configuration can be orchestrated in Playbooks, configuration files written in the YAML syntax. Playbooks can use templates to extend their functionality. Ansible has a collection of modules that can be used to manage various systems as well as cloud infrastructure such as Amazon EC2 and OpenStack. Custom Ansible modules can be written in just about any language, as long as the output of the module is valid JSON. Ansible employs a push-based masterless approach.

Chef has a stable and well-designed layout, and while it is not quite up to the level of Puppet in terms of raw features, it is a very capable solution. Chef may pose the steepest learning curve to administrators who lack significant programming experience, but it could be the most natural fit for development-minded admins. Chef and Puppet are pull-based approach: clients poll a centralized master periodically for updates.

Whereas Puppet and Chef will appeal to developers and development-oriented shops, Ansible is much more attuned to the needs of system administrators. Ansible's simple interface and usability fit right into the sysadmin mindset. Puppet is the most mature and probably the most approachable from a usability standpoint, though a solid knowledge of Ruby is highly recommended. Puppet is the safest bet for heterogeneous environments, but users may find Ansible to be a better fit in a larger or more homogeneous infrastructure.

We choose Ansible for our system, mainly due to its simplicity and gentle learning curve. Unlike Puppet and Chef, Ansible involves only general scripting via the command line. Ansible requires nothing more than a password or SSH key to get started. The user can start managing systems without installing any agent software. Ansible delivers all modules to remote systems and executes tasks, as needed, to enact the desired configuration. Ansible features over 450 modules in the core distribution in which we can tailor to our environment.

B. Deployment Tasks

Our hardware is set up for all services. The slave nodes have been loaded with Red Hat Enterprise Linux Server release 6.7 (Santiago) and proper disk partitioning. A dedicated computer is chosen as the Ansible server. Before getting Ansible running, the environmental variable ANSIBLE_INVENTORY needs to point to a file in which list of hosts is presented and partitioned into host group. Each group is assigned a name, so that group name can be referenced in later operations (e.g., host-slave and host-master). Ansible allows two ways to configure the system, directly calling command ansible with Linux commands, e.g.,

```
ansible <group name> -a ''command''
```

or using ansible-playbook by:

```
ansible-playbook <path to yaml file>
```

The first approach can be used for small tasks and verification of operations, and the second approach can accomplish more complicated task as well as providing better maintainability. The sets of modules are created for each task.

To configure DSP Hadoop cluster in computer nodes and service nodes, there are list of tasks can be done by Ansible:

- create a new user named hadoop
- generate host key and propogate the public key to central authorized key file
- create hadoop/logs and hadoop/pids directory
- rsync packages under /opt to new nodes
- rsync java packages under /usr/local to new nodes
- yum install necessary packages on new nodes
- create /dfs/hadoop and set its ownship to hadoop user

Below is the directory structure for the task *create_logs_pids_dir*,

```
create_logs_pids_dir
    |--roles
        |--create_dir
            |--tasks
                |--main.yml
    |--site.retry
    |--site.yml
```

site.yml specifies target host group, and roles, and the actual tasks will be put in *main.yml* as follows:

```
main.yml
- name: create /var/hadoop/logs
  file: path=/var/hadoop/logs group=hadoop
        owner=hadoop state=directory

- name: create /var/hadoop/pids
  file: path=/var/hadoop/pids group=hadoop
        owner=hadoop state=directory
```

The different tasks will sit in different directories, to allow easier modification and management.

VI. Validation and Tests

We performed a number of tests to evaluate the current clusters' performance and usability. We used four test data sets, summarized in Table I.

IDs	Name	Size
ds1	randu.comb.csv	88GB
ds2	HIGGS.csv	7.5GB
ds3	wiki-Vote.txt	1.0MB
ds4	wikipedia	300GB

Table I
Test Data Sets

Our test cases include running the logistic regression algorithm from Spark ML, executing the classic Wordcount MapReduce program, and evaluating the newly release Spark GraphFrame data structure. Table II shows results of the MapReduce wordcount program executing on the 300GB wikipedia dataset. The program runtimes were collected for various combinations of the memory and heap sizes. Table III shows the result on running Spark ML Linear Regression on dataset ds1, we can see all resources: driver memory, executor memory, number of executor and executor cores contribute to the running time.

map.memory.mb	4096	2048	1560	2560
map.java.opts.-Xmx(MB)	3686	1843	1400	2304
reduce.memory.mb	5120	2048	2048	2560
reduce.java.opts.-Xmx(MB)	4608	1843	1843	2304
Runtime (Hours)	2.18	1.31	1.66	2.11

Table II
Test result of Hadoop Wordcount program running on the 300GB wikipedia dataset.

driver-memory	8G	6G	8G	8G	10G	8G
executor-memory	4G	4G	4G	8G	8G	4G
num-executors	8	4	4	8	8	8
executor-core	4	8	8	4	4	8
Runtime (mins)	27	38	41	49	80	23

Table III
TEST RESULT USING SPARK ML LINEAR REGRESSION ON DS1

The tests were performed to help us better understand how the number of cores contributes to the runtime. To reduce testing time, we used a much smaller dataset ds2 (7.5GB), with the same settings for driver-memory, executor-memory, and the number of executors. The runtime decreases from 3.14min to 1.7min, as number of cores is increased from 1 to 4. This indicates that number of cores contributes positively to performance. Keeping the number of cores constant, changing the driver memory from 4GB to 8GB, the runtime improves from 4.14min to 3.14min.

The small dataset ds3 was used mainly to test that our Spark cluster can support the newly released GraphFrame data structure, designed for graph analysis. We loaded the test data into the system as two DataFrames (Vertex and Edge DataFrame), and then construct a GraphFrame from these two DataFrames. Our test was successful.

These tests only aim to demonstrate that Data Science Platform (DSP) runs normally and as expected. We plan to generate more test cases and use additional large datasets to continue our investigation and tuning of the clusters for different applications and workloads.

VII. CONCLUSIONS

In this paper, we have described our experience in creating a 40-node multi-clusters using donated nodes from Yahoo, built within 2 months. We identified a number of performance, maintenance and sustainability goals. And we detailed our process and thoughtful consideration in choosing configuration tools, hardware partitioning schemes, software stacks; and the optimal solutions are adopted in each category. Our current test suites cover multiple tasks that researchers at Georgia Tech plan to perform using the clusters. Our future work includes bringing up more compute nodes, additional performance tuning, and investigate approaches to support high-throughput communication between clusters.

VIII. ACKNOWLEDGMENTS

This work is supported by the Institute for Data Engineering and Science (IDEaS) [13] Seed Grant. Great thank goes to Partnership for an Advanced Computing Environment (PACE) staffs: Brian MacLeod, Andre McNeil, Dan Zhou, Paul Manno,Josephine Palencia, Blake Fleischer and Michael Brandon for their great support on this work. We thank Daniel Forsyth, Deborah Davis and Meera Kamath for their technical support.

REFERENCES

[1] Amazon emr in details, **https://aws.amazon.com/emr/details/hadoop/**. 2016.

[2] Amazon webservices : Amazon ec2, **https://aws.amazon.com/ec2/**. 2016.

[3] Amazon webservices : Amazon emr, **https://aws.amazon.com/emr/**. 2016.

[4] Ansible:automation for everyone,**https://www.ansible.com/**. 2016.

[5] Apache hbase : A distributed, scalable, big data store, **https://hbase.apache.org/**. 2016.

[6] Apache hive, **https://hive.apache.org/**. 2016.

[7] Apache pig, **https://pig.apache.org/**. 2016.

[8] Apache spark : Lightning-fast cluster computing, **http://spark.apache.org/**. 2016.

[9] Apache zookeeper, **https://zookeeper.apache.org/**. 2016.

[10] Chef — it automation for speed and awesomeness,**https://www.chef.io/chef/**. 2016.

[11] Cloudera impala : analytic mpp database for apache hadoop, **https://www.cloudera.com/products/apache-hadoop/impala.html**. 2016.

[12] Cloudera: The modern platform for data management and analytics, **http://www.cloudera.com**. 2016.

[13] Gt ideas : Georgia tech the institute for data engineering and science, **http://bigdata.gatech.edu/**. 2016.

[14] Hortonworks: Open and connected data platforms, **http://hortonworks.com/**. 2016.

[15] Hortonworks tool set release history, **http://hortonworks.com/products/data-center/hdp/**. 2016.

[16] Hue : Web interface for analyzing data with apache hadoop, **http://gethue.com/**. 2016.

[17] Microsft azure : Hdinsight, **https://azure.microsoft.com/en-us/services/hdinsight/**. 2016.

[18] Opentsdb : The scalable time series database, **http://opentsdb.net/**. 2016.

[19] Pricing for amazon emr and amazon ec2 (on-demand),**https://aws.amazon.com/emr/pricing/**. 2016.

[20] Public cloud comparison matrix, **http://www.cloudamize.com/hubfs/Cloud-Comparison-Matrix.png?t=1471290023181**. 2016.

[21] Puppet open source projects,**https://puppet.com/product/open-source-projects**. 2016.

[22] Welcome to apache hadoop,**http://hadoop.apache.org**. 2016.

[23] Women in hpc 2016, **http://www.womeninhpc.org/women-in-hpc-at-sc16/workshop/submit/**. 2016.