Chapter 8
Big Data Behind Big Data

Elizabeth Bautista, Cary Whitney, and Thomas Davis

Abstract There is data related to the collection and management of big data that is as relevant as the primary datasets being collected, and can itself be very large. In this chapter, we will examine two aspects of High Performance Computing (HPC) data that fall under the category of big data. The first is the collection of HPC environmental data and its analysis. The second is the collection of information on how large datasets are produced by scientific research on HPC systems so that the datasets can be processed efficiently. A team within the computational facility at NERSC created an infrastructure solution to manage and analyze the data related to monitoring of HPC systems. This solution provides a single location for storing the data, which is backed by a scalable and parallel, time-series database. This database is flexible enough such that maintenance on the system does not disrupt the data collection activity.

8.1 Background and Goals of the Project

As more complex computational resources are needed to process, analyze, and simulate large datasets, we need to monitor how these resources are being used; however, the collected monitoring data can also be large and complicated, producing large datasets in itself. Depending on the industry, the monitoring process could be implemented with computers and their associated infrastructure, for example, in robotic systems in the manufacturing lines, in office building facility management systems, and even in wide area network performance or bandwidth monitoring. When the data being collected starts to expand beyond a single device to incorporate information from the surrounding environment and other devices, its complexity increases, leading into the big data realm.

The 4 V's are used to describe big data [1]. This is how the 4 V's map into our problem and the extra element we have to deal with.

E. Bautista (✉) • C. Whitney • T. Davis
National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory (LBNL), USA
e-mail: ejbautista@lbl.gov; clwhitney@lbl.gov; tadavis@lbl.gov

R. Arora (ed.), *Conquering Big Data with High Performance Computing*,
DOI 10.1007/978-3-319-33742-5_8

- Variety (Different forms of data): With the growing concerns with the environment, energy efficiency is becoming a much bigger issue. This data added to computer host and user job data gives a variety of data types, locations and formats that need to be gathered.
- Volume (Scale of data): With the addition of the environmental data and other external considerations have caused the volume of data to increase beyond the basic well-understood issues of computer host monitoring.
- Velocity (Analysis of streaming data): The interplay between a running computer, a global filesystem, which serves the data, and the electrical/mechanical environment that the computer runs in have caused the data collection rates to be increased to help better understand how changes in one system affects the others.
- Veracity (Uncertainty of data): Much of the data collected is uncertain in how it interacts with the data center as a whole. Research work is needed here to determine if the data is needed and at what collection rate.
- Extra element: Our unique element is the different types of consumers desiring knowledge from the data. Each consumer has their own requirements and sometimes, conflict with other consumers. Allowing each individual to gain insights from the same data without too much reconfiguration is a major goal of the project.

In the process of collecting this data, we focused on using open source [2] software in order to provide a framework of a more general scope rather than using software that was more specific in what was being accumulated. As a result, some of the analysis functionality is still in early development. We are providing package names as a reference and starting point to help the reader understand the principles behind this work.

The primary requirements of our collection system came from the needs of our environmental monitoring data and how this information can correlate to the functionality of NERSC's (National Energy Research Scientific Computing Center) new building where the goal is a Power Usage Effectiveness (PUE) [3, 4] under 1.1. (As background information, our computing facility was relocated to a new building where the computation floor that houses the supercomputers and infrastructure systems is cooled by a series of circulating air and water.) PUE is the total power provided to the facility over the power consumed by the computing resources. We wanted to efficiently circulate hot air generated by the computational floor to heat other areas of the building, such as the office floors, and be cooled by air coming from outside the building. This is in addition to the cool water system being circulated through the computational systems. This method and many of these tools could be used in any industry, especially in plant management and services.

8.1.1 The Many Faces of Data

This environmental data collect structure lend to its expansion to the greater NERSC data center serving the Lawrence Berkeley National Laboratory (LBNL), staff,

groups of collaborators, project teams, and collaborators in multi-organizational structures needing data collected to answer specific questions, such as the following:

- How efficiently does the system complete jobs?
- How will the system be affected if we lower the chiller temperature?
- How can we improve the data transfer from one storage library to the computational system?
- How did the power sag affect the processing of long jobs?
- What is the energy efficiency of the facility?
- What is our power utilization on a warm day?
- How does the weather affect a system's job processing?

8.1.2 Data Variety and Location

Some of the data was being collected in areas accessible only for the data owner, while some data was not being collected at all. Because other groups did not have access to this data, and because there was no index of available data, it was possible for another group to duplicate this effort. In addition, the data is in whatever format established by the data owner, which makes it difficult if not impossible for anyone else to reformat the data. We wanted to provide a centralized location and standardized format for this data. This became the primary goal of this project.

At the start of the project, we determined what data was being collected "out there," where they were located, and in what format. As demonstrated by the sample questions above, we learned that the data collected was varied and diverse. It was necessary to provide a dataset that is expandable, adaptable and with a standardized format that one user can use in its original state yet could be reformatted easily by another user for their purpose.

8.1.3 The Different Consumers of the Data

We also considered how the consumers of this data, mainly the management team, the systems administrator, various research experts who assisted in processing the data, the scientific researchers themselves, and our funding agency, would need to interact with the data. Each of these consumers approach data in a different way, and their roles can be illustrated by metaphors of industry. For example, system administrators like plant or facility managers and manufacturing managers, need to know what is happening on the system or computing environment on a moment-by-moment basis. They need information quickly and for a short time. They use this information for fault monitoring, predictive failure, if possible, and performance characteristics.

The management team, like business managers in industry, may need quick and efficient access to the data and any correlations for their various reports to their management or even funding agencies. The research systems analyst wants the immediate view of the data to anticipate issues that may cause researchers any problems when running their jobs or to verify whether a problem may be system-related or not. This individual also needs a longer view of the data to determine if the issues are recurring or if known problems have resurfaced from prior instances. Furthermore, both the system administrator and research systems analyst constantly need to determine if the latest software upgrade has slowed the system's processing speed or has caused some other detrimental effect on the system or if the researcher needs to port his code to sync with the upgrade. If they observe a new outcome that could mean recalculating a new baseline analysis to which new results can be compared.

The scientific researchers are similar to efficiency managers and are interested in system efficiency and performance. The researchers may have written large and complex code to perform computation, to analyze their data, to perform computer simulations or to visualize their data in areas such as climate modeling, fusion research, astrophysics, and genetic research. The models that this code produces tend to be very complex; therefore, optimizing that code could be intricate. In this situation, we want to provide an environment where the researcher can inject messages or statements into their data stream to allow them to measure what is occurring during the computing run to improve speed and performance. These messages should be time-stamped with the same bases as the running job so that the researcher can do correlations. Furthermore, these messages should be easy for the researcher to access but hidden from other users.

Another consumer we had to consider is the system researcher. These individuals usually want the ability to measure how the system environment, such as the filesystem and computing resources, are being used in order to project what the future codes will need. They usually need long-term data so that they can develop or observe trend analysis in both the scientific codes and the computing hardware or software. For example, if software code developers continue to use more computation resources with a smaller memory footprint and to rely more heavily upon the network infrastructure, system architects can incorporate this specification in the next system to purchase. On the other hand, if less computing but more memory resources are needed, the ability to do remote direct memory access (RDMA [5]) in the network may be a required feature. Regardless, we need to provide this data so that these researchers can make a more informed purchase decision on their technical specification and product requirements.

Likewise, the facilities manager needs to know information like whether the new system is using more power or more cooling or if the installed system is using the full rated power draw so that they can make required changes to the building, to capacity planning or plan to correct inefficiencies in the next upgrade. For example, should they choose to spend more money to install 1.5 MW of power because the power supplies indicate that is what is needed when in comparison, the maximum draw of the system is only 0.75 MW? Could this additional expense have been used

at other areas of the facility now? Should that extra power capability be installed now to make future power expansions cheaper? These changes to facility capability can be as drastic as software changes to the scientific researchers.

Lastly, we also examined the needs of the non-consumers—regular users of data whose requirements continued to evolve over time and included the ability to duplicate data to different destinations depending on their usage, move data from one location to another, and save their data in different formats.

8.2 What Big Data Did We Have?

At our organization, we have a diverse research community who has a diverse workload for our computational systems. We have categorized data according to the following five types of data patterns and resources we had observed:

- Collected data
- Data-in-flight
- Data-at-rest
- Data-in-growth
- Event data

8.2.1 Collected Data

Collected data is almost all of the general time-series data. Most of this data will be the environmental data for our new computing facility. This data becomes more important since our center will rely upon the external temperature for the heating and cooling and we need a better understanding of the computing room temperature, especially air circulation. There are five temperature sensors on the front and rear of all the racks, on the floor, including one that goes up the hot chimney that nearly reaches the ceiling. There were also sensors that measured humidity and dust particle count in the air. The new facility uses cooling towers, instead of mechanical cooling, which is a more efficient way of cooling our systems but now we need to measure the flow and temperature of water around the center. The last major data collected in this category is the power flow that included power use measurements and any losses at each step along the way throughout the data center. We measured the power flow from the main transformer through the distribution panels to the Power Distribution Unit (PDU) [6] on the floor and the individual compute nodes, observing that at each of these steps, there was some power loss. We can determine where the major losses have occurred and then devise possible ways to mitigate those losses.

8.2.2 Data-in-Flight

Data-in-flight are datasets created by large experiments and transmitted to various locations for processing. A good example of data-in-flight are the data generated by the Large Hadron Collider (LHC) [7] at the Conseil Européen pour la Recherche Nucléaire (CERN) [8], in Geneva, Switzerland. The CERN beamline can generate up to one petabyte of data per day and send that out to as many as 170 different research institutions for processing. This requires knowledge of how the wide area networks function and, more important, of the filesystem where the data will come to rest. An important aspect of transferring these large datasets is knowing the filesystem's write characteristics and expected performance. This helps to determine if the filesystem is sufficient to accept the data and if it has the ability to recover should there be any interruptions in the data flow across the wide area network. For example, while this data is being transferred "in flight," we need to collect data on network speeds, latency, and packet loss all along the wide area network to ensure a clear path from starting point to destination. In addition, as data is being received by the filesystem, we are collecting data on the disk write speeds, I/O, CRC errors [9], etc., to ensure a successful receipt.

8.2.3 Data-at-Rest

Data-at-rest are large datasets that researchers conduct searches against. This type of data will experience many reads from many different systems that could result in data hot spots. A hot spot occurs when many processes want to access the same data and all these processes could be slightly out of sync with each other, causing the physical disk head to seek around the disk, thus causing performance problems for all the processes. There are strategies to mitigate this problem, ranging from caching to data duplication, but first one needs to know where, when, and to what extent the problem is happening.

8.2.4 Data-in-growth

Another data type we have is data-in-growth where we observe many processes generating data from an initial set. This is slightly different from data-in-flight where we have to ensure the filesystem write capability is able to accept the data. In the case of data-in-growth, we need to collect data to ensure that writes can be processed rapidly from multiple compute hosts so that they do not impact the application's performance.

8.2.5 Event Data

The last data type we have is event data. Events need to be correlated with the running jobs since many of them will affect jobs in either a good or bad way. Some of this data comes from data normally collected by the system such as the following:

- Syslog
- Console logs
- Program checkpoints
- Hardware failure events
- Power events
- Anything that has a start and end time

8.2.6 Data Types to Collect

With the five defined data types and their resource requirements, we are now able to determine a set of data points to be gathered for the collector.

- Computing performance and system health data on the host(s): This included data about CPUs, memory, networking counters, user and some system processes, filesystem counters, and overall node health.
- Network performance: Data may be collected from the hosts but we also need to collect data and statistics from the switches to find oversubscribed network links and to determine the health of the switches themselves.
- Filesystem data: the biggest and most diverse type. Depending on the filesystem type, we collect the used and maximum size of the partitions to monitor for filled disk partitions or servers. We also collect data on the filesystem server load: Is one system being used more heavily than the others? If yes, why and where is it coming from? We use the read and write performance data to tune the filesystem to read/write block size, stream characteristics, and to determine applicable file sizes. Each of the above data types tends to have different filesystem needs.
- Environmental data: ambient temperature, humidity, particle counts, power, or water flow temperature. These sensors may require additional networking and infrastructure to bring the data back to the central collector.
- Event data: This type of collected data can be one of the trickiest to manage because most event data logs are easily accessible but difficult because it contains private information.

8.3 The Old Method Prompts a New Solution

Given that some of this data was already there, we wanted to know how the users at NERSC were collecting it, and what process was used to interact with this data. We had to implement many different tools over time to get the data we needed. Although many of these tools were very good at doing a specific task, they could not be expanded to a more general role. Like a beloved car that you grew up with, you eventually outgrew it or your needs changed. That was the case for most of our old monitoring and data collection system.

8.3.1 Environmental Data

The environmental data consisted of a mired of protocols and very old software with many custom data collections methods. In addition, one software package had a limitation of only functioning on a Windows 98 system using obsolete hardware, where the collection rate was one data point every 5 min. This method did not give us the resolution or the response time we needed for the new facility. Examples of the protocols used are the following:

- SEAbus™ [10]—The Siemens Energy & Automation, Inc two-wire communication protocol over RS485 for power management.
- Modbus™ [11]—The Modbus Organization serial communication protocol between electronic devices.
- Modbus™/RTU—Modbus™ protocol over serial communication like RS485 or RS232.
- Modbus™/TCP—Modbus™ protocol over TCP/IP.
- SNMP [12]—Simple Network Management Protocol. Used to collect data and program devices over TCP/IP.
- BACnet® [13]—An American Society of Heating, Refrigerating and Air-Conditioning Engineers standards protocol for building automation and control networks.
- Metasys® [14]—A Johnson Controls protocol for building automation and control.
- Johnson Controls—A building automation system.
- Metasys2—Allows non-Johnson Control devices to participate in a Metasys® network.
- Incom™ [15]—Eaton Corporation two-way INdustrial COMmunications between a network master and Cutler-Hammer breakers and switches.

We tried to create a common datastore in MySQL™ [16], of the Oracle Corporation for the environmental data. MySQL™ offered an easy to install and configure solution, and thus offered a low barrier to entry while still providing a

powerful mechanism to data access. The problems that we started to encounter were when the database became very large, in excess of six billion items. At this point, malformed queries caused the database server to become unresponsive and hang.

We alleviated these issues by upgrading MySQL™, getting a bigger system, and tuning the configuration, but again we soon hit the limit. We realized at that point that MySQL™ was not an easy solution, and we soon required a full-time Database Administrator (DBA) to manage the database. However, even maintaining the database became a chore. Backups either took too long or could not be completed because of table-locking issues. When the hardware failed, we lost hours of uncollected data because it took too long to restore. The process of pruning the database involved creating a new table and dropping the old, which is a quick but inefficient approach when you are trying to recover the old data and transferring it to a different machine.

In summary, MySQL™ met some of the desired goals of the project, such as developing a database that offered a single location, a standard format, and ease of use for multiple users but it did not meet our other goals to achieve a resilient, interchangeable, composable [17], and easy-to-maintain system. At this point, we decided we needed a fresh and innovative solution and relocating to the new building offered a clean break from the old processes.

8.3.2 Host Based Data

Our staple monitoring tool, Nagios® [18], has served us well. With its updated user interface and refreshed data collection methodology, we expect to continue to use Nagios® in other areas of the facility. However, the package is unable to collect data over a period of time. For example, Nagios® alerts us about a filesystem being full but not about how fast the disk partition filled up over time.

The second tool we used was Ganglia [19], which collected many of the host-based metrics that we required and offered the ability to add additional metrics via a plugin environment. However, its default datastore—Round Robin Datastore (RRD) [20]—emphasized speed and size over data volume, which did not lend itself to long-term analysis. High-resolution data was summarized into lower-resolution data over time. While Ganglia's Graphical User Interface (GUI) provided some correlation functionality, it is actually optimized for the system administrator, not necessarily for any other user.

We also used the Lustre® [21] Management Tool (LMT [22]), a specialized monitoring application. LMT's default datastore, MySQL™, gave us some difficulty that we experienced with our other MySQL™ databases. We also needed to collect additional information from these nodes, and doing so meant we would have to run two or more data collectors, a process we had no desire to implement.

Homegrown scripts are another group of monitoring tools that needed to be replaced. These scripts directly addressed needs that came about during the

operation of the systems. Most of these scripts suffer from both the consistent storage of data as well as the problematic display of the results. Most of the time, only the author and some other initiate knew how to use and interpret the data.

8.3.3 Refinement of the Goal

To overcome the challenges of our old methods, we propose a new collection system that:

- Is centrally located outside of any computational system. One previous method was to collect the data on one of the compute nodes. Managing this as a separate entity means the hardware can be retired and upgraded when needed.
- Is in a common storage format for all data to ensure that future data analysis can function seamlessly.
    - Enables individual monitoring components to be changed out when new or better solutions become available.
    - Is flexible enough to add new data collections and processing as we learn more about the data we are collecting.
    - Is secure enough to protect system data while allowing researchers access to discover new interactions in this ecosystem.
    - Allows private data associated with a single job and intended for one particular researcher.

8.4 Out with the Old, in with the New Design

In the new facility, we wanted to address limiting the number of protocols in the environmental data collection area. To do this, we standardized on a common PDU in the computing rack setup, and built a monitoring network into the infrastructure that allowed the collection of the environmental data to be more consistent and on a reliable timescale.

From the data collection point of view, we needed to break this problem into its primary parts:

- Data collection
- Data transport components
- Data storage
- Visualization and analysis

An all-encompassing program that would carry out all parts of the monitoring process would not work. We knew that such programs start out well, but over time, grow to be overwhelming to maintain, and that all system flexibility would be lost. We therefore settled on an Advanced Message Queuing Protocol (AMQP)

[23] solution used by RabbitMQ® of Pivotal Software, Inc. [24]. The overall result is a scheme that allows any component to be swapped out or replaced when better or more appropriate solutions become available. By achieving modularity, the packages that we selected to use in our solution could be replaced with other similar packages to achieve the same results.

8.4.1 Elastic

The ELK stack by Elastic [25] stands for Elasticsearch, Logstash, and Kibana, an integrated solution that helps analyze the data in real time. Each of these products is open source and well supported. Used together, it provided us the tools we needed to search through and analyze our data. Elastic provides the ELK stack as a commercially supported product with some extra features, which we are evaluating at this time. Elastic has now provided Beats, which is a collection method for system metrics and logs.

With Elastic providing some form of the each of our requirement, we still needed to supplement their offering.

Elastic, Beats and Filebeat are trademarks of Elasticsearch BV.

Elasticsearch, Logstash and Kibana are trademarks of Elasticsearch BV, registered in the U.S. and in other countries.

8.4.2 Data Collection

8.4.2.1 Collectd

The major data collecting component of the framework is Collectd [26]. Like other data collection tools, Collectd has plugins for reading and writing the data and allows many different configurations to choose from. The design goal is if Collectd needs to run on a node to collect data, it should be the only thing running to collect data. We sometimes found that special purpose gateways may run data collection methods needed for a specific environment, and that those methods would send their data directly into the monitoring framework. Collectd offers the following benefits:

- A consistent data structure and definition to the information being collected.
- Since the data is sent via UDP, problems in the network or infrastructure do not take out computing resources. We do not want the monitoring to cause a job to fail. Thus the aggregation point takes most of the brunt of the monitoring process.
- Plugins can be written in C [27], even though there are Perl and PHP hooks.
- Data collection rates and plugins can be configured independently.

8.4.2.2 Custom Scripts

Our search for open source solutions did not always work, thus many of the environmental sensors required a custom solution. We tried to keep it simple by creating a common framework for the collection framework and this framework fed into RabbitMQ®.

8.4.2.3 Filebeats

Filebeats is the Elastic offering that follows log files and forwards them to a different destination. We are using it to capture much of our event data, sending it into data collect structure.

8.4.3 Data Transport Components

8.4.3.1 RabbitMQ®

RabbitMQ® provided the NERSC facility the following benefits:

- A fault-tolerant distribution setup.
- A rich number of plugins to connect to other applications.
- Accepts a good number of input applications.
- Offers a buffering capacity if segments of the infrastructure go down through the use of durable queues.
- Smooths out the bursty [28] profile of the collection process.
- Easily add data readers to a stream.
- Adds security via encrypted channels.

8.4.3.2 Logstash

We had considered using Logstash as the main connectivity between components, but RabbitMQ® has a few more benefits, and both applications work very well together. Logstash is used as the feeder into Elastic and is also the link between the remote systems. A diagram of the overall layout is shown in Fig. 8.1.

Logstash, when used as the local system aggregation point, gives us some very important functions:

- Reduces the number of network connections that the central logging clusters need to manage. The system's local server takes the local connections and forwards a single connection to the center logger.

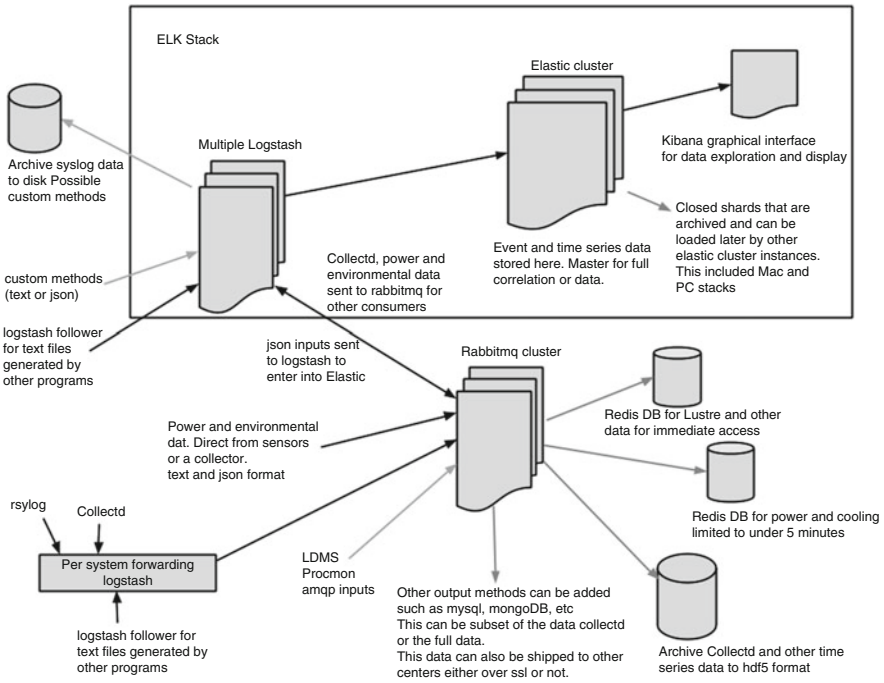

Fig. 8.1 Central Data Collect data flow

- Provides the encryption point so non-encrypted data is only present on the local system or within the local system where the data may be protected by other methods.
- In conjunction with the Collectd method, converts the UDP [29] packets that Collectd uses to TCP as a way to keep from dropping packets and loosing data. Collectd is further explained in its own section.

Logstash is also used for these applications:

- The collection and forwarding of text logs. These include syslog, console logs, and other text-based logs. Logstash has the ability to follow hard-to-forward text files, which allows us to scrape files like the torque accounting record file, console output files, etc. This file data is also forwarded to the central logger over a single stream.
- To provide a method for user applications to insert data into the collection path. Having the application write a JSON [30] output line and sending it to logstash to accomplish this.

8.4.4 Data Storage

8.4.4.1 Elasticsearch

Elasticsearch is the datastore for the ELK stack. Elasticsearch is easily clustered, has a nice shard structure, and has many different add-on applications available. Elasticsearch offers the following benefits:

- New nodes can be added to the Elasticsearch cluster without downtime to the entire cluster, and the datastore will begin to rebalance and reallocate seamlessly.
- Add-on applications for shard management and archiving.
- Add-on to allow datastore access to look similar to a SQL database.
- Since Elasticsearch is written in Java® of Oracle Corporation, it can run almost anywhere, meaning researchers who would like to analyze long-term data correlations could get individual shards and load them onto their laptops.

8.4.5 Visualization and Analysis

8.4.5.1 Kibana

The last component of the ELK stack is Kibana, the visualization tool of the stack that provides the ability to generate dashboards of the data. Kibana offers the following features:

- Integrated access to Elasticsearch datastore.
- Intelligent data delivery to the browser. For example, if the graph shows 5-min data increments and the data was collected at 1-s increments, an average will be sent to the browser instead of the full raw data stream.
- Other types of data visualization beyond the standard bar, line, and pie charts.

Another major design component of our framework is the stream processing. RabbitMQ® can write a copy of a subset of the data directly to the consumer. This is accomplished via the STOMP [31] protocol and will forward to different applications for display purposes. This allows the following:

- On-the-fly processing of the data. The only load time would be the data collection interval of the longest data point.
- Stream anomaly detection via applications such as Heka [32].
- Display dashboard such as Freeboard [33].

8.4.6 *Future Growth and Enhancements*

Lastly, with the ability of RabbitMQ® to republish copies and subsets of the data stream, we can load that data into other applications that can answer the questions some of our researchers have of the system. These include:

- A stream of data to a consumer to archive all data to HDF5 [34] format, which allows researchers to use a language such as R [35] effectively in correlating events.
- Streaming a subset of data to MySQL™ to populate an existing database with the same data that was gathered by different means so websites and other portals continue to function seamlessly.
- A subset of data could be streamed to a process that could mimic writing to the RRD files of a Ganglia setup to allow the GUI to be viewed by those who find it useful.
- A Redis® a trademark of Salvatore Sanfilippo [36] datastore could be populated with filesystem data for a month, allowing support staff fast access to the latest performance data and thus helping system troubleshooting.
- A second Redis® datastore with tombstone data: With only one value stored, the old value is dropped once a new value is added. This allows applications such as Nagios® to query this datastore for some of its check information instead of connecting to a host and running a program.

8.5 Data Collected

The largest part of any monitoring project is the data being collected. We are collecting from sensors in different areas of the center to perform correlations between all types of events that could affect the running of the center and the jobs running on the computing resources. We collect the following types of data:

- Electrical
- Environment
- Building automation
- Host
- Application
- Events

8.5.1 Environmental

Electrical

We are collecting data from sensors at the substation, through the different levels of PDU's available in the building, down to the node level if possible. This also includes the UPS/generator setup. Figure 8.2 below shows how data is collected from the subpanels and passed to the central monitoring cluster via a Power Over Ethernet (PoE) [37] network setup. The PoE setup is needed since many of these switches and panels are located far away from the traditional network infrastructure.

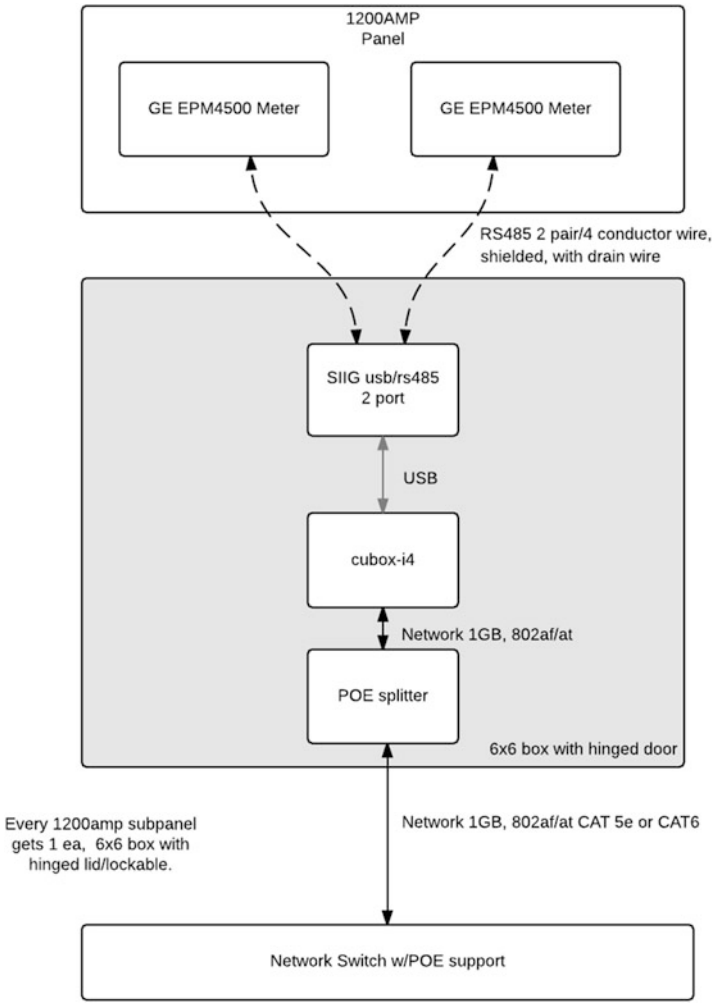

Fig. 8.2 PoE switch diagram

For example, the substation might not include normal 110 power for the sensors and hardware, which is why PoE is an easy way to get both networking and power to these remote locations. We estimate we will need 3000+ sensors with each sensor collecting 10 data points with a frequency of 1 s.

Environment
These are the sensors that collect water flows and humidity readings, and measure temperatures in different areas of the racks and center. Many of these sensors are used to ensure that the computing floor is functioning correctly to keep the systems from experiencing an issue. For example, on each rack door, there is a one-wire [38] network of 12 temperature sensors per rack, with four on each door and two on top. There are also 22 sensors in the ceiling and several under the floor. The 5000+ sensors will collect a single data point every 20 s.

Building Automation
These sensors tend to be everywhere except the HPC floor. This is primarily the lights and heating/cooling in which the latter is controlled by the workload on the computing floor. This is the smallest sensor suite but probably an interesting component to measure how the systems operate. Here we are planning for 500+ sensors collecting over 50 data points per sensor at a frequency of 2 s.

Most of the electrical, environmental, and building automation are sent into the monitoring cluster via glue code and gateways into RabbitMQ®.

8.5.2 Computational

Host
This is data we collect from the computational and support systems. The computation pool is over 11,000 nodes and may only get a summary treatment for data collection. This is one area that is still under discussion since the collection of data can slow the computation of an application. Our primary function is to serve the scientific research community, not hinder it. The computing arena is one area where computation power is at its premium.

The second type of hosts is the filesystem servers, login nodes, gateway nodes, and other types of support nodes. These nodes will have a base collection of network statistics, mounted filesystem size, memory usage, process count, users logged in, and some other general items. With a minimum of 3500+ sensors and upwards of 150 data points per sensor, this is one of the largest single sources of data. This data will be collected at a frequency of 5 s but will change to 1 s for research purposes.

A sample listing of data collected would be:

- 1-, 5- and 15-min load average
- CPU utilization
- df, diskstats of the local filesystems

- The IP network interface statistics
- The InfiniBand® from the InfiniBand Trade Association network statistics
- Processes running on the system
- Users who are logged in
- Swap and memory usage.
- Uptime

Application
One of the major applications that run on all systems is the filesystem. Here we are measuring the two global filesystems at the facility: Lustre® and GPFS™ of International Business Machine Corporation (IBM) [39]. Most of these filesystems have two different components: the server side and the clients. There will be some collection of data from both sides, but primarily the server side will be targeted first. Since the filesystems are very complicated, they tend have much more data per collection cycle. For example, the Lustre® filesystem has 500+ sensors collecting data points at 5-s collection intervals and 3500 data points per sensor. We are not measuring GPFS™ since it is still in development.

The collected data is gathered from the proc [40] filesystem and includes all the MDS [41] and OSS [42] servers:

- filesfree, filestotal
- kbytesavail, kbytesfree, kbytestotal
- md_stats
- stats
- job_stats
- brw_stats

One aspect of the collection environment allows us to enable collection of data when needed. The Lustre® client data can be collected, but we are not doing so at this time. This would require 100+ sensors collecting data at 5-s intervals and equivalent to 2500 data points per sensor. This could include the 11,000 compute nodes and much of the same data being collected in the paragraph above but from the client's point of view.

8.5.3 Event

Events
Another major datastore is syslog and other types of log data. This is slightly different from the time series performance data that is always considered but very important in determining the health and events of the system and center. This data gives us the start and stop time of jobs; critical and sometimes nonfatal errors from hardware; security data for incident detection; and other events to annotate the time-series data. This can add up to 200–300+ events per second.

The last data component for applications is the researcher-generated data, which include items such as library usage, modules loaded, license checkouts, different events that may be desired to help diagnose issues that may arise on the systems, and user-defined program checkpoints to help measure the job's progress. These events are inserted or converted to the JSON format for easy insertion into Elastic.

Since all the sensors have not been installed yet, we have tested this layout in other ways. We have loaded filesystem metadata into the system at rates upward of 350,000 inserts per second from a single process along with the three or four other input streams and three output streams without the database being affected. We have also had over six billion items in the Elasticsearch datastore. Dashboard accesses have been rather fast, and with further tuning, we do not see any issue there. Lastly, this process uses less physical storage than in our old method, approximately 1 terabyte.

8.6 The Analytics of It All: It Just Works!

Let's start by watching a data flow. In this particular case, we will start with Collectd and the paths that its data will take.

Collectd is running on our system "X," which is a Lustre® MDS for one of our systems. This filesystem is named "scratch." To start Collectd, run the following command as shown in Table 8.1.

Inside the configuration file (collectd.conf) we have something that looks like the paragraph in Table 8.2.

Things to note here is that the types.db file that contains all the data definitions for the data Collectd gathers is also shared by Logstash, which decodes the binary Collectd data stream and converts it to JSON.

Table 8.1 Running Collectd from the command line

./collectd -C collectd.conf

Table 8.2 Beginning of the collectd's configuration file collectd.conf

BaseDir	"/opt/collectd"
PIDFile	"/tmp/collectd.pid"
PluginDir	"/opt/collectd/lib/collectd"
TypesDB	"/etc/logstash/types.db"
Include	"/opt/collectd/etc/collectd.d/base.conf"
Include	"/opt/collectd/etc/collectd.d/mds.conf"
Interval	10
ReadThreads	25
WriteThreads	25

Table 8.3 Common elements of the collectd.conf file. These will be read from the base.conf file

base.conf
LoadPlugin cpu
LoadPlugin df
LoadPlugin disk
LoadPlugin interface
LoadPlugin load
LoadPlugin memory
LoadPlugin network
LoadPlugin processes
LoadPlugin swap
LoadPlugin uptime
LoadPlugin users
LoadPlugin vmem
<Plugin df>
MountPoint "/global"
IgnoreSelected true
ReportInodes true
</Plugin>
<Plugin network>
Server "logstash-local.domain.com" "25826"
MaxPacketSize 1024
</Plugin>

Table 8.4 Environment specific configuration for the collectd.conf file. Read from the mds.conf file

mds.conf
LoadPlugin mds
LoadPlugin ib

The two included configuration files in Table 8.3 (base.conf) and Table 8.4 (mds.conf), contain the needed plugins and are specialized based on node function.

When we load all of our plugins, the df plugin tells Collectd not to collect df information for the filesystem mounted at /global and to report inode information. While the mds.conf file tells Collectd to load the two special plugins: one for the Lustre® MDS, which collects the Lustre® MDS statistics and the other, collects Infiniband® network statistics.

The network plugin is used to send binary data from this Collectd instance to "logstash-local.domain.com," which is the local instance of Logstash. It is then Logstash's job to forward the data into the center-wide monitoring environment.

Now we move the local Logstash. Its job is to forward local data to the center-wide collector. Thus it is started on the host with a sample name of 'logstash-local.domain.com' that has a configuration file similar to Table 8.5:

What does all this do? First, when it receives a Collectd packet, it decodes it with the associated types.db files and then adds some tagging information to give Elastic and downstream processors a bit more information (see Table 8.6). Then it adds two

Table 8.5 The logstash configuration file

```
input {
  udp {
    port => 25826
    buffer_size => 1452
    workers => 3
    queue_size => 3000
    codec => collectd {
      typesdb => [ "/etc/logstash/types.db", "/etc/logstash/local-types.db" ]
    }
    tags => [ "collectd", "X" ]
    add_field => { "system" => "X" }
    add_field => { "function" => "scratch" }
    type => "collectd"
  }
}
input {
  tcp {
    port => 1514
    type => "syslog-X"
    tags => [ "syslogd", "tcp", "X", "internal" ]
    add_field => { "system" => "X" }
  }
  udp {
    port => 1514
    type => "syslog-X"
    tags => [ "syslogd", "udp", "X", "internal" ]
    add_field => { "system" => "X" }
  }
}
output {
  if [type] == "collectd" {
    rabbitmq {
      host => "rabbit-host.domain.com"
      port => 5671
      ssl => true
      verify_ssl => false
      exchange => "ha-metric"
      exchange_type => "topic"
      durable => true
      persistent => false
      user => "<RabbitMQ user>"
      password => "<Password>"
      key => "%{system}.%{function}.%{host}.%{type}.%{plugin}"
    }
  } else {
```

Table 8.5 (continued)

```
    rabbitmq {
        host => "rabbit-host.domain.com"
        port => 5671
        ssl => true
        verify_ssl => false
        exchange => "ha-log"
        exchange_type => "topic"
        durable => true
        persistent => false
        user => "<RabbitMQ user>"
        password => "<Password>"
        key => "%{system}.%{host}.%{type}"
    }
  }
}
```

Table 8.6 Logstash configuration file for the receiving logstash process. This creates the RabbitMQ® exchange

```
rabbitmq {
  host => "rabbit-host.domain.com"
  port => 5671
  ssl => true
  verify_ssl => false
  exchange => "ha-metric"
  queue => "ha-system-metric2elastic"
  user => "<RabbitMQ user>"
  password => "<Password>"
  key => "system.#"
  prefetch_count => 1500
  threads => 3
  codec => "json"
}
```

fields that are treated as hard information. This is more than just a tag. These fields are used later on when sending the data to RabbitMQ®. This instance of Logstash will also forward syslog messages.

Once the packet comes in and is processed, Logstash will compare it to the output types. Once it finds a match, it sends the packet using an encrypted connection to the RabbitMQ® cluster. In this case, the Collectd stanza will be used, and the data will be sent with the AMQP key, "%{system}.%{function}.%{host}.%{type}.%{plugin}" or 'X.scratch.mds-host.collectd.{mds,load,user,vmem,..}' This information is sent to the RabbitMQ® queue ha.metric.

From there, RabbitMQ® forwards ha.metric messages based on the %system key. For this data, the system key is "X," and RabbitMQ® will forward this to ha.X.metric. If ha.X.metric exchange has not been create, RabbitMQ® will then create it.

From here, there is a Logstash listening on queue connected to ha.X.metric and forwarding all incoming data to the Elastic cluster. This queue is sectioned off based on the system key passed in from the collection side. If the queue does not exist, it will be created and linked to the listed exchange. The 'key' field will be used to subscribe to the type of data coming from that exchange.

There could also be a Freeboard instance listening in on ha.X.metric via the STOMP protocol, but it could be asking for data only from "X.scratch.*.collectd.load," which would give this Freeboard all the load averaged for all the hosts in the function group scratch.

Lastly, a feed from that same queue could go to a Redis® datastore to keep the MDS data for the next 30 days. However, this would be only a portion of the data and not the full data flow.

Now, we go back to the Logstash listener on ha.X.metric. This process takes the data streaming to it and places it into Elastic for long-term storage and dashboard analysis.

This is how we implemented Elastic and how it is configured:

- All data is collected into location-event shards, which are created on a daily basis. In this example: X.function.collectd.
- The most recent shard of data less than 24 h old is stored on Solid State Drive (SSD) [43] attached locally to each Elasticsearch node.
- Two-day-old shards are migrated from the SSD drive to local drives also on each node. This data is still available to the Elasticsearch datastore.
- At this point, the two-day-old shards are archived to the global filesystem. Researchers now have the ability to load these shards back into their personal copies of the ELK stack.
- This data is copied to our long-term tape storage for archival purposes.
- After 30 days, this data is removed from the local drives and the Elasticsearch database.
- Data index management is performed by the Knapsack [44] Elasticsearch plugin.

The SSD step was needed to help Elasticsearch keep up with the data rates we are approaching. This will also be an "ingest-only" Elasticsearch, meaning that most uses of the data will be on an external Elasticsearch system which would insulate the data collecting cluster from inappropriate queries.

After the data has been ingested into Elasticsearch, Kibana now has access to it. At this point, Kibana dashboards can be created to display parts of the collected data. We set the data source for Kibana to be "X.function.collectd." We then followed the on-screen menus to explore and create graphs. Below are examples of such a dashboards.

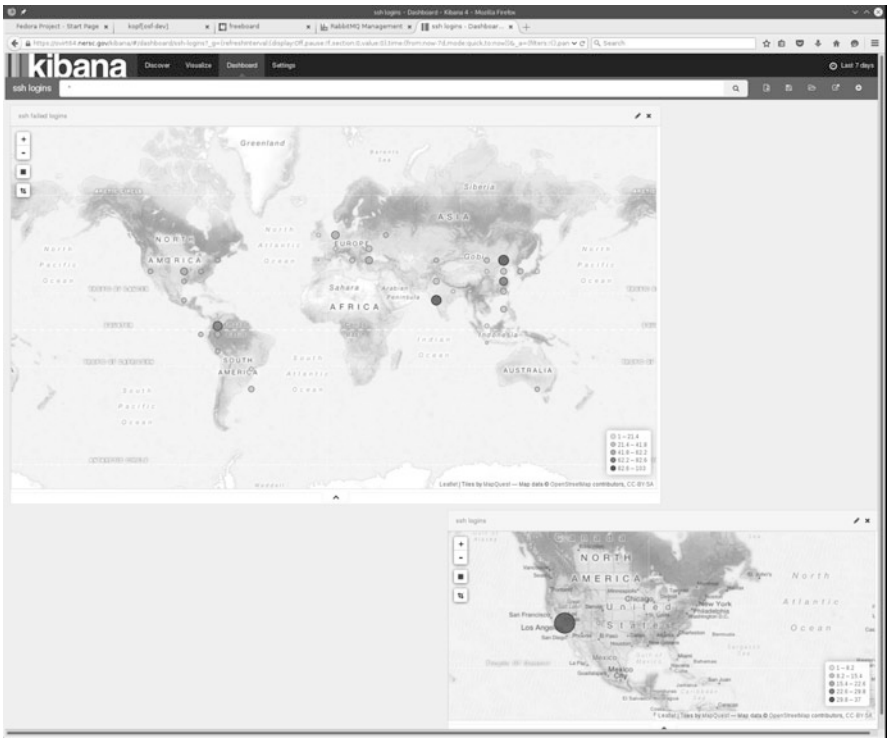

Fig. 8.3 Failed ssh connection locations

This Kibana dashboard (Fig. 8.3) shows the failed ssh [45] connection attempt into some of the cluster nodes. Other visualizations we could show could be the computer room heat map, resource utilization for network bandwidth, disks, computational resources, or other spatial data. The following Kibana dashboard shows typical line graphs of some performance metrics (Fig. 8.4).

The last part of the framework is the cluster management. We are using oVirt™ [46] of Red Hat, Inc to manage the virtual environments. All the Logstash and RabbitMQ® instances are running in oVirt™, which allows us to migrate an instance to another node if we need to perform any cluster maintenance. Another advantage of the virtual environment is we can turn up additional Logstash and RabbitMQ® nodes if we need them for special processing. The Elasticsearch nodes are all running on bare metal [47] since the virtual environment does not give us much benefit. Elasticsearch itself does the replication and failover for its cluster.

Fig. 8.4 General network statistics graphs

8.7 Conclusion

Building a monitoring infrastructure is exciting, challenging, and a necessity given our computational environment and the growing needs for efficiency and understanding in many other environments. While there is no single way to do this, it still needs to be done. The biggest issue of collecting monitoring data is choosing what data to collect. You need to know what questions can be answered by monitoring; thus if you know the question, you know what to collect. However, you can get into a very complex discussion of what data to collect, what format you can put it in, and how much to collect. This thinking leads to the fear of collecting too much data or data that is not needed. Another thought is to collect everything but that also leads to inefficiencies of collecting too much.

In our scenario, we have taken a hybrid approach that collects the data that was already being collected in some fashion, and supplements it with some data points that we think we will need. A more important consideration for us is to have a solution that allowed us to collect data in a single location was resilient, interchangeable, and composable. As a result, we were able to provide a database that is flexible when inserting new data as well as editing or archiving data, yet

flexible enough that we can actually perform maintenance on the system without disrupting the collection activity.

Finally, we have talked about monitoring data in a supporting role for other sciences, but many of the techniques and information presented could be used for primary collections. Data is data: it is our frame of mind that puts meaning to that data.

References

1. IBM Big Data & Analytics Hub (2016), http://www.ibmbigdatahub.com/infographic/four-vs-big-data. Accessed 8 Feb 2016
2. Open Source Initiative (2016), http://opensource.org/. Accessed 8 Feb 2016
3. The Green Grid (2016), http://www.thegreengrid.org. Accessed 8 Feb 2016
4. Wikipedia, Power usage effectiveness (2016), https://en.wikipedia.org/wiki/Power_usage_effectiveness. Accessed 8 Feb 2016
5. Wikipedia, Remote direct memory access (2016), https://en.wikipedia.org/wiki/Remote_direct_memory_access. Accessed 8 Feb 2016
6. Wikipedia, Power Distribution Units (2016), https://en.wikipedia.org/wiki/Power_distribution_unit. Accessed 8 Feb 2016
7. Large Hadron Collider (2016), http://home.cern/topics/large-hadron-collider. Accessed 8 Feb 2016
8. CERN (2016), http://home.cern/. Accessed 8 Feb 2016
9. Wikipedia, Cyclic redundancy check (2016), https://en.wikipedia.org/wiki/Cyclic_redundancy_check. Accessed 8 Feb 2016
10. Siemens Energy & Automation, Inc, 4720 SEAbus protocol reference manual (1995), http://w3.usa.siemens.com/us/internet-dms/btlv/ACCESS/ACCESS/Docs/4720_SEAbusProtocol_Reference.pdf
11. Wikipedia, Modbus (2016), https://en.wikipedia.org/wiki/Modbus. Accessed 8 Feb 2016
12. Wikipedia, Simple network management protocol (2016), https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol. Accessed 8 Feb 2016
13. BACnet (2016), http://www.bacnet.org/. Accessed 8 Feb 2016
14. Johnson Controls, Metasys (2016), http://cgproducts.johnsoncontrols.com/default.aspx?topframe.aspx&0. Accessed 8 Feb 2016
15. D. Loucks, INCOM protocol documentation, EATON (2015), http://pps2.com/smf/index.php?topic=7.0. Accessed 8 Feb 2016
16. MySQL™ (2016), https://www.mysql.com/. Accessed 8 Feb 2016
17. Wikipedia, Composability (2016), https://en.wikipedia.org/wiki/Composability. Accessed 8 Feb 2016
18. Nagios® (2016), https://www.nagios.org/. Accessed 8 Feb 2016
19. Ganglia Monitoring System (2016), http://ganglia.sourceforge.net/. Accessed 8 Feb 2016
20. T. Oetiker, RRDtool (2014), http://oss.oetiker.ch/rrdtool/. Accessed 8 Feb 2016
21. Lustre® (2016), http://lustre.org/. Accessed 8 Feb 2016
22. LMT (2013), https://github.com/chaos/lmt/wiki. Accessed 8 Feb 2016
23. AMQP advanced message queuing protocol (2016), https://www.amqp.org/. Accessed 8 Feb 2016
24. RabbitMQ (2016), https://www.rabbitmq.com/. Accessed 8 Feb 2016
25. Elastic (2016), https://www.elastic.co/. Accessed 8 Feb 2016
26. Collectd (2016), https://collectd.org/. Accessed 8 Feb 2016
27. Wikipedia, Programming language (2016), https://en.wikipedia.org/wiki/Programming_language. Accessed 8 Feb 2016

28. Wikipedia, Burst transmission (2016), https://en.wikipedia.org/wiki/Burst_transmission. Accessed 8 Feb 2016
29. Wikipedia, User datagram protocol (2016), https://en.wikipedia.org/wiki/User_Datagram_Protocol. Accessed 8 Feb 2016
30. JSON (2016), http://www.json.org/. Accessed 8 Feb 2016
31. STOMP (2016), https://stomp.github.io/. Accessed 8 Feb 2016
32. Heka (2016), https://hekad.readthedocs.org/en/v0.10.0b1/. Accessed 8 Feb 2016
33. Freeboard (2016), https://freeboard.io/. Accessed 8 Feb 2016
34. The HDF5 Group (2016), https://www.hdfgroup.org/HDF5/. Accessed 8 Feb 2016
35. Wikipedia, R (programming language) (2016), https://en.wikipedia.org/wiki/R_(programming_language). Accessed 8 Feb 2016
36. Redis® (2016), http://redis.io/. Accessed 8 Feb 2016
37. Wikipedia, Power over Ethernet (2016), https://en.wikipedia.org/wiki/Power_over_Ethernet. Accessed 8 Feb 2016
38. Wikipedia, 1-Wire (2016), https://en.wikipedia.org/wiki/1-Wire. Accessed 8 Feb 2016
39. IBM, IBM Knowledge Center (2016), http://www-01.ibm.com/support/knowledgecenter/SSFKCN/gpfs_welcome.html?lang=en. Accessed 8 Feb 2016
40. Wikipedia, Procfs (2016), https://en.wikipedia.org/wiki/Procfs. Accessed 8 Feb 2016
41. Lustre® MDS (MetaData Server) (2016), https://build.hpdd.intel.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.xhtml#understandinglustre.tab.storagerequire. Accessed 8 Feb 2016
42. Lustre® OSS (Object Storage Server) (2016), https://build.hpdd.intel.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.xhtml#understandinglustre.tab.storagerequire. Accessed 8 Feb 2016
43. Wikipedia, Solid-state drive (2016), https://en.wikipedia.org/wiki/Solid-state_drive. Accessed 8 Feb 2016
44. Knapsack (2016), https://github.com/jprante/elasticsearch-knapsack. Accessed 8 Feb 2016
45. OpenSSH (2016), http://www.openssh.com/. Accessed 8 Feb 2016
46. oVirt™ (2016), http://www.ovirt.org/Home. Accessed 8 Feb 2016
47. Wikipedia, Bare machine (2016), https://en.wikipedia.org/wiki/Bare_machine. Accessed 8 Feb 2016