MACHINE LEARNING BASED DATACENTER MONITORING FRAMEWORK

by

RAVNEET SINGH SIDHU

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2016

To my family

Acknowledgements

I would like to express my sincere gratitude to Mr. David Levine my thesis supervisor who has guided and motivated me in the course of this thesis. His continuous support with patience and enthusiasm has helped me in contributing my research work in this thesis.

I would also like to thank Mr. Patrick McGuigan, my mentor for his help, support and guidance throughout my research at (High Energy Physics) HEP's datacenter for ATLAS experiment. It was his support, continuous feedback and invaluable suggestions that have helped me at each stage during this research work.

I would like to acknowledge and thank my committee members, Dr. Ramez Elmasri and Dr. John Robb for their co-operation, feedback and support.

My heartfelt thanks to my mother for always motivating me to do the best, my father for supporting me in chasing my dreams and my sister for encouraging me in times of despair. Special thanks to Jasmine Grewal and Abhinav Bansal for their continuous support and encouragement. Finally, I would like to thank all my friends who have helped me in this journey.

November 11, 2016

Abstract

MACHINE LEARNING BASED DATACENTER MONITORING FRAMEWORK

RAVNEET SINGH SIDHU, M.S.

The University of Texas at Arlington, 2016

Supervising Professor: David Levine

Monitoring the health of large data centers is a major concern with the ever-increasing demand of grid/cloud computing and the higher need of computational power. In a High Performance Computing (HPC) environment, the need to maintain high availability makes monitoring tasks and hardware more daunting and demanding. As data centers grow it becomes hard to manage the complex interactions between different systems. Many open source systems have been implemented which give specific state of any individual machine using Nagios, Ganglia or Torque monitoring software.

In this work we focus on the detection and prediction of data center anomalies by using a machine learning based approach. We present the idea of using monitoring data from multiple monitoring solutions and formulating a single high dimensional vector based model, which further is fed into a machine-learning algorithm. In this approach we will find patterns and associations among the different attributes of a data center, which remain hidden in the single system context. The use of disparate monitoring systems in conjunction will give a holistic view of the cluster with an increase in the probability of finding critical issues before they occur as well as alert the system administrator.

.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

1.1 Introduction and background

In our growth and technology centric world, we are adding petabytes of data per day. To process this information in a fast and reliable manner we rely on Grid Computing or Cloud Computing. In the world of High Performance Computing (HPC), service availability has great importance. In order to maintain high service availability with growing number of data centers, the need for efficient monitoring solution is of great importance.

Monitoring is a generic term in the context of data center health check. It can have a very different view depending upon the various components in a data center. Person A might be concerned about job running status, person B wants to see the number of servers which are working, person C need to keep a check on the cluster load and other network related metrics. In order to cover all the different requirements a general approach followed is to implement different monitoring solutions based on each use case and there is no single monitoring solution, which can cover end-to-end flows. This solution increases the overall complexity of monitoring the health of datacenters, as it adds the number of alerts to be monitored and the number of different scenarios an operations person has to monitor. Tones of alerts are triggered from these systems and all monitoring systems maintain their own dashboards. Each system may also send email notifications to the system administrator to update about possible issues. All the alerts and dashboards act as a point of source of information into the health of a datacenter. The data collected measures some important and also some less important information about the actual health of the datacenter, for example some of the physical attributes like total disk space used may not be as interesting as swap space used when considering runtime health check of datacenter running high load jobs.

In terms of categorizing and differentiating the importance of some of the attributes reported by monitoring solutions requires a much harder and time-consuming effort. A system

administrator is required to first understand all the attributes in detail and then derive correlation among the different parameters to mitigate failures. Things become complicated when the administrator has to analyze data from disparate monitoring solutions and wants to provide insightful patterns about the health of a datacenter.

The need of a machine learning based datacenter monitoring framework is eminent in order to improve the service availability and drive better understanding about the impact of different attributes reported by disparate monitoring solutions. It will also enable the logging of system data, which can act as single source of truth. We can do leverage the use of a single platform for analyzing anomalous patterns, which pertain to failure and cause system crashes. The framework can help us in better understanding the system usage patterns such as the CPU memory consumption, data generation and inbound/ outbound network traffic details. Studying these details with the help of machine learning can help us in finding anomalies in this system which maybe harder to find just focusing on the dashboards or alerts sent by different monitoring solutions. We can also leverage machine learning to provide a single context to the monitoring problem and connect all the input monitoring channels to a single engine, which could then be extended in order to have a holistic view of the system and understand patterns from across the monitoring solutions.

In our work at UTA we support the ATLAS [1] data center, which we will discuss in the next chapter in details, where we primary use three different open-source monitoring solutions as described below and see Figure 1-1 for an overview of the architecture.

- We use Nagios [2] to monitor servers, services on server, switches and alerting system.
- Ganglia [3] keep a check on all the machines in respect to usage and network load.
- Torque [4] is used to manage job scheduling and keep track of resource utilization.

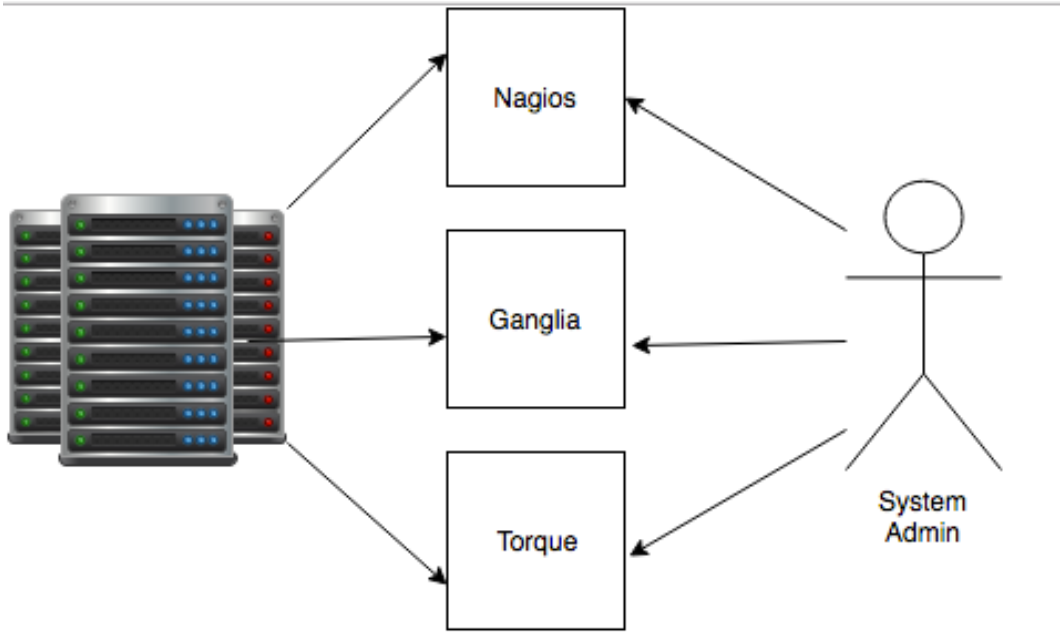


Figure 1-1 ATLAS Datacenter Monitoring

In UTA's ATLAS [1] datacenter we have more than 300 compute nodes, which are managed with the help of the three monitoring solutions as described previously. To find anomalous nodes among these set of machines is a challenging task. Each solution has a different set of attribute measurements, which cover different aspects inside the datacenter. There can many different patterns among the attributes that can cause failure. A higher CPU time spent on input/output can be one and nodes going into swap space can be another. It becomes very difficult to categories bad nodes based on single attribute without having a clear line of separation.

In this work, a machine learning based datacenter monitoring framework is presented. A single solution with this concatenated approach is taken to provide an overall measure of the datacenter health and performance. As there is no scale to compare the bad vs. good nodes, we leverage the use of an unsupervised learning model known as clustering. This analysis helps in identifying similar working nodes and identifying the outliers in the dataset. Due to

concatenation, the dataset becomes high dimensional and it becomes very hard for us to analyze the results. We use Principal Component Analysis (PCA) in order to reduce the dimensionality by minimizing the information lost during reduction. We will be discussing the machine learning algorithms used in coming chapters.

1.2 Motivation behind the thesis

The monitoring of datacenter is a daunting task, as it requires a lot of time and human effort. The most efficient solutions only cover some part of the monitoring problem and we have to use a combination of solutions that are best suited for this case. The monitoring of a set of solutions generating a plethora of system alerts, statuses or warnings is an unrealistic task for any system administrator. Even by analyzing visual representation of information generated at runtime, it's challenging to carve out solid informatory correlations among the attributes. Without having an in-place monitoring system in order to derive valuable actionable information from the dataset, the probability of improving the efficiency of datacenter is reduced. In order to assist, the need to upgrade the monitoring solutions without performing any changes in an already practiced and proven solution, the idea of developing an abstracted machine learning based monitoring system on the top of all the monitoring solutions is established.

A machine learning based system will be able to maintain state of all the attributes associated with the nodes within the cluster. It can help in discovery of some of the problems that remain unseen due to the velocity, variety and volume of data being generated by all the monitoring systems. The detection of anomalous nodes that are not performing at scale can help us save a lot of time and money. Figuring out a high number of nodes having frequent issues such as memory problem, disk IO errors can prevent wastage in terms of resource utilization. A holistic view of the datacenter can also detect correlations among different attributes from disparate monitoring solution and also give an insight into dependent issues that may go unseen by a single monitoring solution. As we maintain the overall state of all machines,

we can find time sensitive patterns associated with the load. It can help us in better estimating the required computational power and we can add or subtract resources as needed. This will further help us in getting rid of non-performing resources and saving money by increasing utilization.

1.3 Goals of thesis

The main objective of this thesis is to implement machine learning based data monitoring framework for finding anomalous patterns by analyzing a single high dimensional view of data from disparate monitoring solutions deployed on a datacenter. In order to use the existing monitoring solutions that are covering all the monitoring aspects of the datacenter, an abstraction layer is created on top of the existing framework. The state of all nodes is analyzed by quantifying the attributes with the help of machine learning algorithms and any possible outliers are marked.

Information regarding possible outliers helps us in better understanding about the health of datacenter and also facilitate us to action that further impacts the overall performance of system.

1.4 Organization of thesis

The thesis in Chapter 1 start with an introduction and background about the topics, it also covers the motivation and goals of the work. Chapter 2 gives an overview of related work. Chapter 3 dives into an insight into the ATLAS experiment at CERN and current approach. Chapter 4 provides the background of the problem statement. Chapter 5 describes the initial experiment and flaws found in the approach. Chapter 6 covers the detailed description of the machine learning framework - bunny. Chapter 7 concludes this thesis with summary of the thesis. Chapter 8 outlines the future work.

Chapter 2

Related Work

In this era of technological advancements, the task of gathering and managing the data created around the globe has become of utmost importance. It is estimated by that we add about 2.5 quintillion bytes of data [5] per day. In order to manage and process this volume of data efficiently higher computational power is needed. To meet the high demand, we leverage the use of high-performance computing, which can be described as the use of parallel programming for running processes. The addition of HPCs clusters to manage data intensive tasks is also heavily adding large amount of resources to the pipelines. The management of these resources in the most efficient manner is a daunting task as it adds to the complexity of the overall system. Extensive research has been conducted on the management of data center or HPCs to formulate best practices and develop methods that provide an overall insight into the health, service availability and performance of the datacenter. In this work we study related research methodologies used by different researchers and follow principals in order to implement our custom solution.

2.1 Dimensionality of Data

The management of a high performance computing machine is a complex issue, as a quantitative check needs to be maintained on a significant number of machines with different aspects of the datacenter. Knobbe et al. [6] discusses the complexity of a big enterprise network monitoring due to the interdependence of several attributes in causing service degradation. It presents the idea of using different data mining approaches by applying them on time-sensitve data collected from network based monitoring solutions. The first experiment describes file system performance by collecting data with an interval of 30 minutes, measuring response time and resulting a data set of 5052 records. It takes into account following attributes like File Host, Directory Size, CPU Usage, and Response Time. The second experiment discusses network

bottlenecks by collecting data from a single router across 16 subnets over a period of 18 weeks and totaling 16849 measurements on each subnet. Knobbe et al. [6] uses bandwidth utilization, packet throughput and number of collisions across each subnet. It also describes an experiment using real time data of a spare part tracking and tracing application for aircraft. It studies the causes that affect the change of behavior of performance metrics. The data collected was high dimensional with 250 parameters at an interval of 15 minutes for duration of 2 months and resulting in a table of 3500 time slices of 250 parameters totaling the data with 875000 entries. It uses attributes such as CPU load, free memory, database reads, and NFS activity. Data-mining techniques it uses are decisions trees, top n algorithms, rule induction algorithms and inductive logic programming.

2.2 Centralized System

The use of a centralized system is important to maintain the state of the entire data center. Renesse at el. [7] describes a scalable distributed management system termed Astrolabe. Astrolabe is used to monitor the dynamically changing state of distributes resources and then report the summary of utilization patterns to its users. It organizes the resources into hierarchy of zones and associates attributes with each zone. The system is designed to poll chunks of information to retrieve the state of the resources in the network and also performs data aggregations at runtime to enable timely access of the system information. Aggregation is used as a centralized mechanism to perform various tasks based on the bounded scope so that even in case of the worst condition, the size of system is not impacting in the performance of overall system. The scope is bounded for each aggregate and the results are visible only to the nodes in that particular scope.

Astrolabe acts as agents for collecting and disseminating information about the zones. A zone is either a host or a set of zones that don’t have a host in common. All zones are uniquely

identified by a zone name, which is a string consisting of its path of zone from the root. It maintains the hierarchy throughout the system by forming a path separated by slashes. If not specified by the administrator the zones are formed in a decentralized manner following the path. Each host runs an agent of Astrolabe and the leader from within a set of hosts are elected to take responsibility for running gossip protocols that maintain these internal zones if anyone of them fails or becomes unsuitable, an other host will be selected by the protocol automatically. Aggregation functions are used within each zone to calculate the attributes at zone level, which is a SQL script, taking the list of all attributes and forming a report for a particular zone. Astrolabe, by design is a relatively small object, which can range from few hundred to even thousands of bytes. The small size and the scale at which they can be deployed make them usable to be used efficiently in a distributed environment. Apart from these features, Astrolabe also brings power with simplicity and flexibility to the distributed environment that make it usable in configuration management problems like peer-to-peer caching of large objects over distributed applications which operate on single big files. It also finds its use in multicasting of games across the globe to the users without putting a TCP-unfriendly load on the Internet. The multicasting feature as described in the case of peer to peer can also be used to unable publish-subscribe system without having a hard limit on the number of messages.

2.3 Abstraction Layer

The use of different systems to monitor all the aspects related to health of datacenters is needed. As each solution is developed to solve a particular use case in an efficient manner, we need to also develop a way to improve the services by pro-actively looking for issues. Issariyapat et al. [8] focuses on a better usability of Nagios as a network monitoring system by proposing suggestions in order to improve the product and increase the overall utilization. It outlines some of the core weaknesses in the system design such as system configurations are based on a complicated text file that make it harder for administrators to change or find the

issues, the enabling of logging of more sensor data such as temperature, addition of work flows by using the Nagios Event Broker (NEB) into the core system design and integrating advanced web technologies by using CSS and AJAX to improve the older CGI based web client of Nagios.

The architecture described in the study strongly aligns with the idea of not altering the Nagios source code but formulating plugins or directing outer environment variables to the added layer of functionality. The modular architecture of Nagios core enables the proposed system design to be easily compatible with the future release of the open source version. The design focuses on the use of custom variable macros to tackle the problem of unique identifier in the system. A clear distinction is placed to separate the custom variables by using an underscore sign in front. In order to enable logging of information in a database a new plugin-broker named Nexec is developed which places itself above all other plugins and executes all other scripts on behalf of core Nagios. Once it is triggered, it gathers results from all plugins, stores the results to the database and also sends the results back to Nagios.

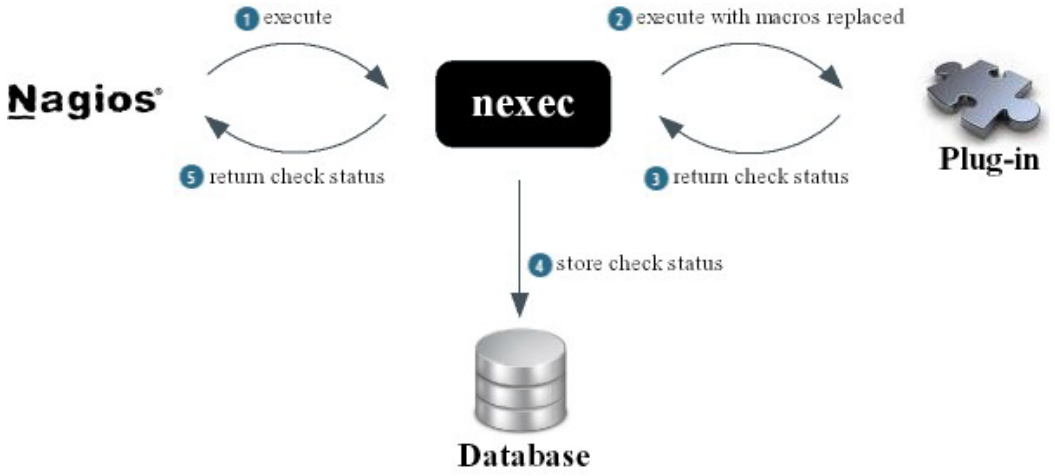


Figure 2-1 Updated architecture as described in Issariyapat at el. [8]

The Network topology provided by Nagios is not real time and to upgrade the functionality an external user interface module is implemented using flash. The topology of nodes connected to

the system is maintained and proper coloring is enabled to assist a user in figuring out the problematic nodes at glance. Figure 2-1 provides an overview into the architectural design.

2.4 Machine Learning Models

In order to drawn correlations from multiple attributes in a high dimensional space, we cannot just depend on the use of graphs and visualizations. The use of machine learning models is a right approach. Sahoo at el. [9] focuses on the concept of autonomous computing where the system maintains a self-check on its current health and is also able to self-heal from various failures. It also stores monitoring information about the state of all nodes in order to analyze patterns and minimize future system failures by using machine learning prediction models. The key issues of cluster system such as reliability, availability and serviceability are analyzed by using real time generated by 250-node cluster. Using the analytical results from the system events after processing them through multiple machine learning and artificial intelligent engines. A self-managed and control system is proposed based on time series, classification for prediction and root cause analysis through Bayesian networks.

Chapter 3

An Insight into ATLAS and current approach

3.1 Understanding ATLAS

What is ATLAS?

As explained in Wikipedia [10], “ATLAS stands for (A Toroidal LHC Apparatus), is one of the seven particle detector experiments constructed at Large Hadron Collider (LHC) [11], a particle accelerator at CERN [12]”. During the collision of protons high energy is released which results petabytes of information per experiment. ATLAS helps in discovery of new theories of particle physics, which are beyond the Standard Model [13]. It will help us answer the questions related to origin of our universe, extra dimensions of space, microscopic black holes and evidence for the presence of dark matter in the universe.

How much data is created?

As per the ATLAS website [14], if all the data is recorded from the experiment would be recorded it would fill 100 000 CDs per second. It would create a stack of CDs which would be 150 m (450 Ft.) high every second. , It could reach to the moon and back twice each year. In terms of data size it record 37% of whole data produced during experiment.

How is such huge amount of data is processed?

The generated data from CERN is transferred at the rate of 1 GB/s to 10 main computing centers in 3 continents. From here the stored data is processed and further distributed to another 50 collaborating institutes for analysis. It analyzes 15 petabytes (15 million gigabytes) of data generated every year generated by the Large Hadron Collider (LHC) [15].

3.2 Architecture of Worldwide LHC Computing Grid

The data from the experiment is distributed across the world with primary data being collected and stored at CERN [15]. After processing the data, it is being distributed globally to eleven big data centers with large storage space, computing power that also has round the clock support for this grid. These eleven data centers make data available to the over 120 other center for analysis of tasks.

The grid system is divided into “Tiers” which distribute data across the globe in order to help in processing. In total it has three ‘Tiers” involving 32 countries.

- Tier-0: This is the main site called the CERN computing center where all the data first reaches and is processed out to other next Tier. All data goes through Tier 0 first but it has only 20% of computational capacity.
- Tier-1: It is comprised of the eleven large data centers across the globe, having high compute and storage power. These data centers are located in Canada, France, Germany, Italy, the Netherlands, the Nordic countries, Spain, Taipei, and the UK, with two sites in the USA.
- Tier-2: This group is a collection of 140 sites which are grouped into 38 federations covering Australia, Belgium, Canada, China, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Hungary, Italy, India, Israel, Japan, Republic of Korea, the Netherlands, Norway, Pakistan, Poland, Portugal, Romania, Russia, Slovenia, Spain, Sweden, Switzerland, Taipei, Turkey, the U.K, Ukraine, and the U.S. Tier-2 sites will provide around 50% of the capacity needed to process the LHC data.

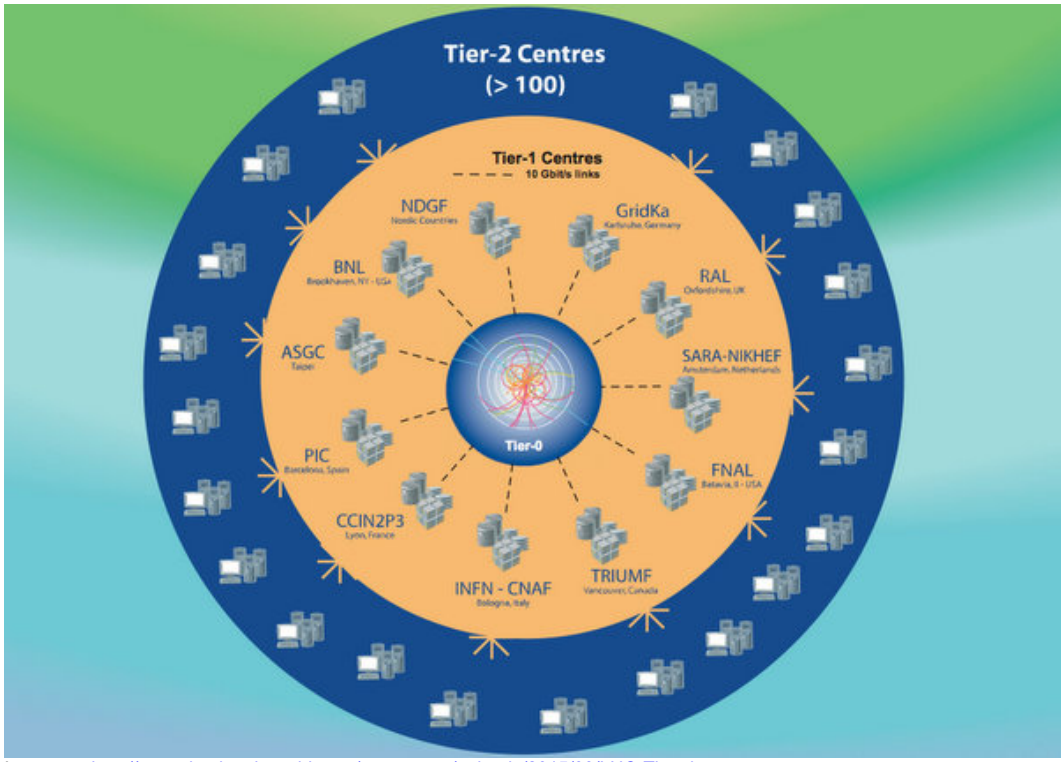


Image src http://www.theskepticsguide.org/wp-content/uploads/2015/03/LHC-Tiers.jpg

Figure 3-1 CERN Grid Architecture

3.3 ATLAS Data Center at UTA

The data center hosted at UTA for providing computational and storage resources in order to help the ATLAS experiment is a Tier-2 center. In terms of internal cluster design, it is based on master slave architecture. Primarily it has two types of nodes, namely compute node and storage node. Compute nodes are responsible for handling computation part of the workload, which is submitted to the system. It has 350 compute nodes that provide a strong backbone to the job scheduling system. The storage nodes provide data storing capacity to the data center, which are primarily being storing data, received from CERN [12] experiments and jobs. There are currently around 21 Nodes present in the datacenter which add to ~3 Petabytes storage to the system. Apart from the primary node types there are special nodes being used

for performing particular tasks such as running special jobs and for cluster maintenance, which are the AAS (American Astronomy Society) and XRDB (X resource database manager) [16].

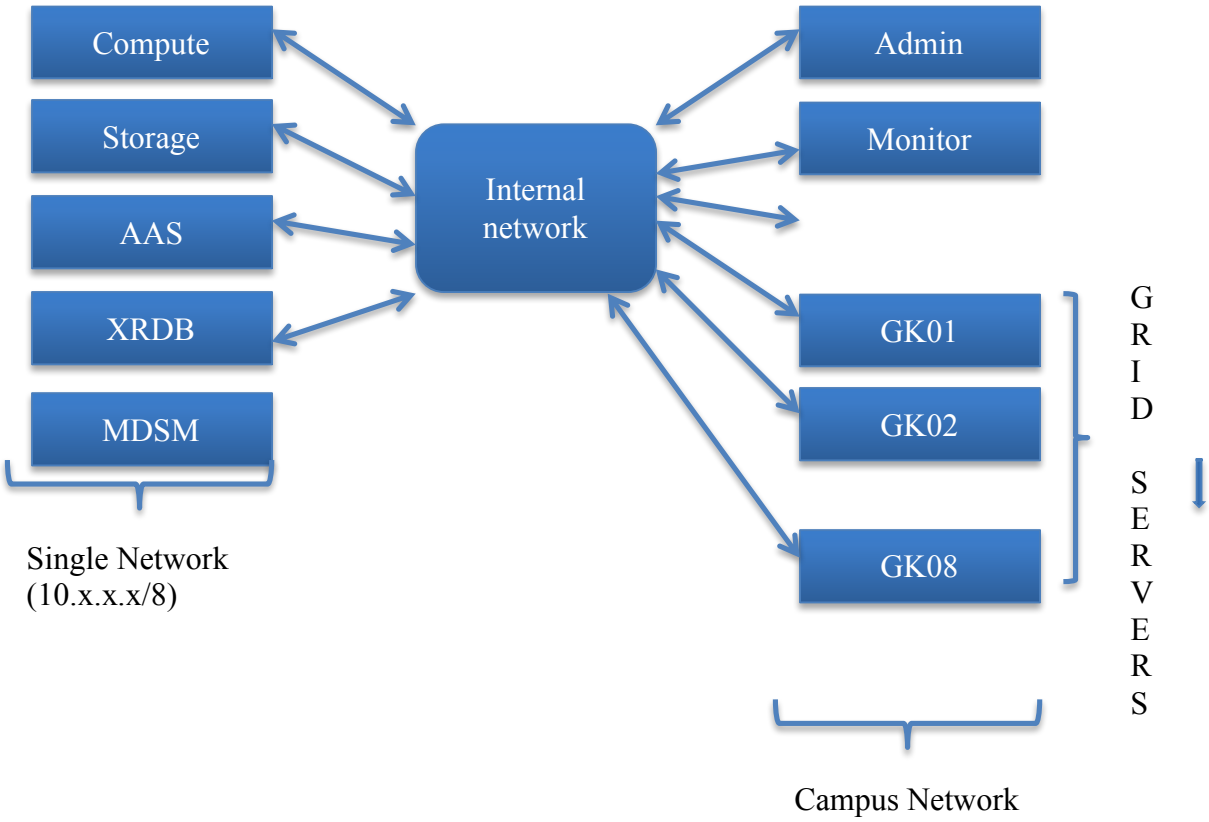


Figure 3-2 ATLAS Datacenter at UTA

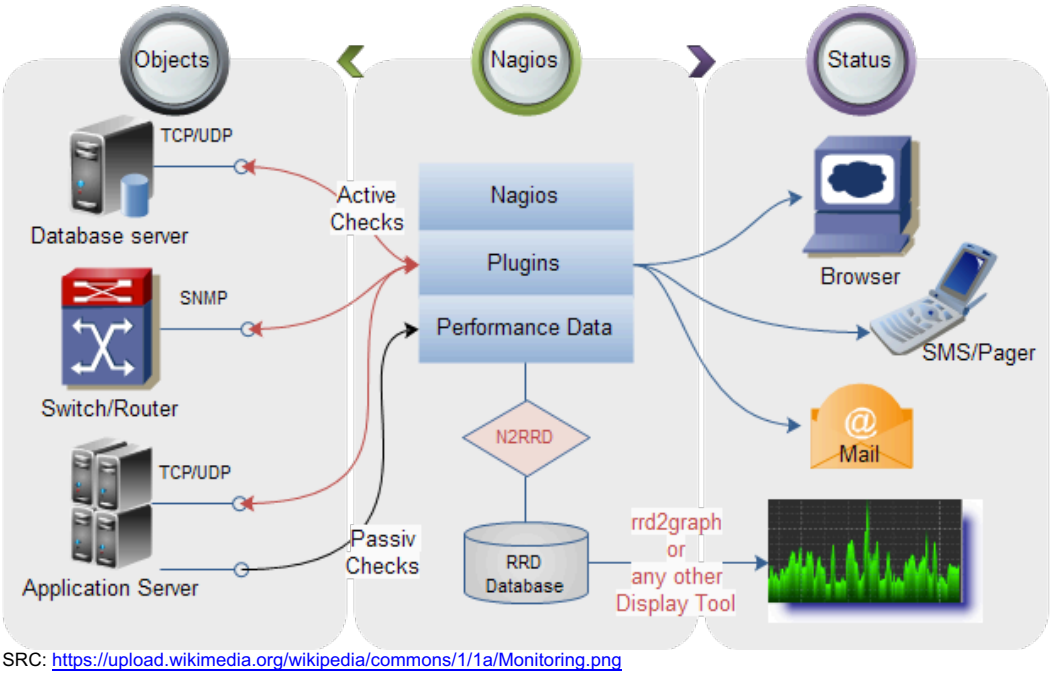
3.4 Monitoring Services

In order to manage the data center a combination of different monitoring solutions have been deployed on all the nodes of the datacenter. They cover all aspects of data center monitoring. The combination used at the datacenter includes Nagios, Ganglia and Torque as described in the following section.

3.4.1 Nagios

Nagios® [2] is an open source system and network monitoring application. It watches the hosts and services that have been specified to it and alerts on problems and then when the problems have been resolved. Nagios can monitor network services (SMTP, POP3, HTTP, NNTP, PING), and host resources (processor load, disk usage). It supports monitoring of Windows, Linux/Unix, routers, switches, firewalls, printers, services, and applications. Its plugin design allows users to develop their own service checks. Nagios supports defining network host hierarchy using "parent" hosts, allowing detection of and distinction between hosts that are down and those that are unreachable.

Users can define event handlers to be run during service or host events for proactive problem resolution. There is an optional web interface for viewing current network status, notification and problem history, log file, etc. Nagios can automatically restart failed applications, services and hosts with event handlers. Nagios scales to monitor over 100,000 nodes and has failover protection capabilities.



SRC: https://upload.wikimedia.org/wikipedia/commons/1/1a/Monitoring.png

Figure 3-3 Nagios Monitoring

Nagios uses its agents to collect different types of metrics on each node and collects the information to present in form of alerts/subscriptions. It has agents running on all nodes within the datacenter that report the metrics information to the master node, which concatenates data and reports. One of the main agents is NRPE (Nagios Remote Plugin Executor). It allows remote system monitoring using scripts hosted on nodes, it also allows getting metrics such as disk usage, swapping usage, CPU Usage. The information is then collected and stored on to RRDs (Round Robin Database)

3.4.2 Ganglia

Ganglia [3] is a distributed monitoring system for High performance computing (HPC) systems. It is used to manage large clusters, networks and grid computing infrastructures. It gives an insight into the real time and historical view of different metrics such as network load,

CPU load. It uses XML for data representation, XDR (External Data Representation) for portable data transport and RRDs data storage and visualization. It is a passive system, where it waits for machines to report state to the master node. It [17] is made up of mainly four components as described below:

- Gmond (Ganglia monitoring daemon): a service that collects information about a node and is installed on all servers.
- Gmetad (Ganglia meta daemon): a daemon on the Monitor node that collects data from all the Gmond daemons.
- RRD (Round Robin Database) tool: a tool on the Monitor node used to store data and visualizations for Ganglia in time series.
- PHP web front-end: a web interface on the Monitor node that displays graphs and metrics from data in the RRD tool.

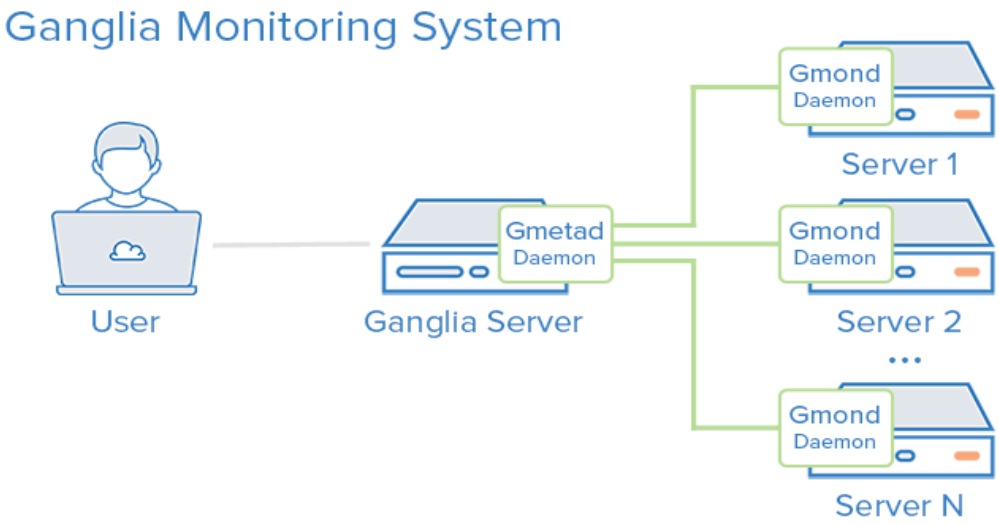


Image SRC: https://assets.digitalocean.com/articles/ganglia_ubuntu14/1.png

Figure 3-4 Ganglia Architecture

3.4.3 Torque

Terascale Open-source Resource and QUEue Manager (TORQUE) [18] [19] is a distributed resource manager providing control over batch jobs and distributed compute nodes. It has been extended from PBS (Portable Batch System) [20] to include scalability, fault tolerance, scheduling and usability. TORQUE server contains all the information about a cluster. It knows about all of the MOM (Machine Oriented Mini-server) nodes in the cluster. It also maintains the status of each MOM node through updates from the MOMs in the cluster. All jobs are submitted via qsub to the server, which maintains a master database of all jobs and their states. It is responsible for scheduling jobs received from different sources. It assigns jobs to different queues depending upon the type of job and requested load. It maintains many queues and some of the important ones are mentioned below:

- Default
- Atlas analysis
- Multi core
- Atlas Prod

3.5 Current Approach

A snapshot of data aggregated by different monitoring solution is presented as an HTML page with legends as visible in Fig [3-5]. The page is refreshed each 5 minutes by the script and each time the current view is presented. The report covers all the racks with different servers stacked as per the physical setup. Each node has labels that are color-coded based on monitoring status of all the 3 monitoring solutions.

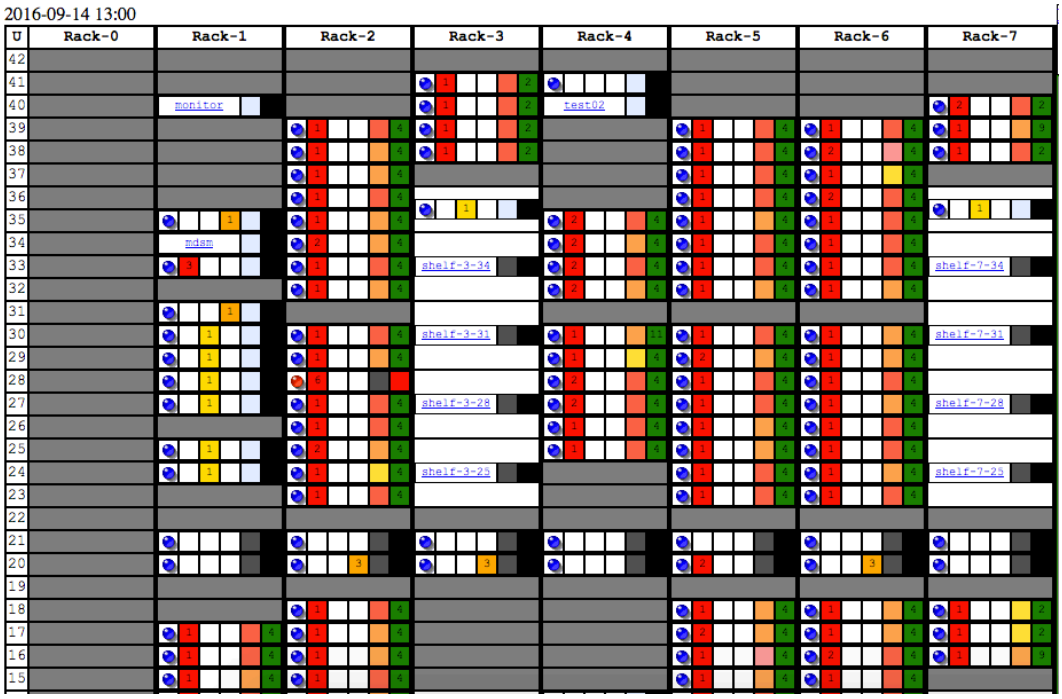


Figure 3-5 Current Snapshot of Cluster

3.5.1 Understanding Node Legends

A node's state is normally shown with six boxes indicating various state conditions learned from Nagios, Ganglia, and Torque. In the table above, the boxes are labeled as P, A, W, U, G, and T. The first four represent information from Nagios, while the last two represent Ganglia and Torque, respectively. Figure 3-6 gives a detailed snapshot of all the states.

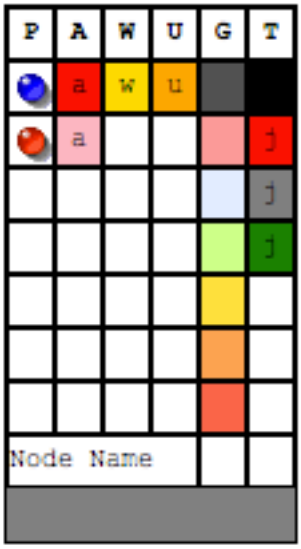


Figure 3-6 Snapshot of states

The P column refers to the Node's state as defined by Nagios and it means that the host can be pinged. A blue ball indicates that Nagios considers the node to be up. A red ball indicates that Nagios considers the node to be down.

The A column refers the number of services associated with the node in Nagios that are considered to have an alert status. The column can be empty, red with a number, or pink with a number. An empty square indicates that there are no services in alert status. A red square with a number indicates that there are services in alert status and one or more are unacknowledged. A pink square with a number indicates that there are services in alert status and all of the alerts are acknowledged.

The W column refers the number of services associated with the node in Nagios that are considered to have a warning status. The column can be either empty or yellow with a number. An empty square indicates that there are no services with a warning status. A yellow square with a number indicates that number of services have a warning status.

The U column refers to the number of services associated with the node in Nagios that are considered to have an unknown status. The column can be either empty or orange with a number. An empty square indicates that there are no services in Unknown status. An orange square with a number indicates that number of services have an Unknown status. If given node is not known to the Nagios system, the four columns (P,A,W,U) will be replaced with a single box containing the node's short name.

The Ganglia status is displayed under the G column. The column shows either grey, pink, or various other colors based on the machine's load. A grey square indicates that the node in unknown to Ganglia and the node's name does not exist in the gmond output. A pink square indicates that the node is known to Ganglia but has not been heard from in more than three minutes. The other colors available to the Ganglia status indicate that the machine has recent Ganglia data. It uses ‘load_one’ (Load in per minute) metric to compute the color. The exact color used is based on the calculation of int (((load_one / num_processors) * 4)+1) yielding a number between 1 and 5 where 1 represents an idle node (color=light blue) and 5 represents a fully loaded machine (color=red).

The last column, T, represents the status of the machine in Torque and is shown with a number. The indication colors are: black, red, grey and green. The number indicates the number of jobs currently running on the machine. A black color indicates that the node is not a Torque compute node and has no status. A red color indicates that the node is considered to be down in Torque. A grey color indicates that the node is considered to be offline in Torque with the indicated number of jobs. A green color indicates that the node is in a normal state for Torque. The last row of the table above indicates a blank U in the rack.

Chapter 4

Problem Statement

In order to perform monitoring, we need substantial manual effort to understand the current health of the data center. From time to time, a dedicated resource is required to keep monitoring the health of datacenter. We do use email notifications, which are sent via Nagios but these are generally overlooked due to the bulk of email hitting the email box each 5 minutes. The monitoring data is stored for Ganglia in the RDDs but not for other monitoring solutions.

Due to the velocity and variety of data coming from monitoring solutions, it is difficult to draw any correlation among the attributes. The absence of single source of truth (data) makes it even harder to perform any kind of analytics. We need a system, which can help us in better understanding of correlation among different attributes and also allow an approach to be able to perform analytics.

Chapter 5

Initial Analysis

5.1 Log Files

In the initial approach, we used generated log files from monitoring solutions as a source of metrics. The logged data available to us was for duration of 3 months. The concept was to design a system, which would read from log files, extract useful indicators from different sources and then store them in a database. The new data source then would be cleaned, to remove any missing values. Again the data would be scaled to bring numeric value in the same range and then feed into an un-supervised machine learning algorithms to perform analysis of data. Visualization would then be used to understand the data plots. Figure 5-1 highlights the architecture used in this solution.

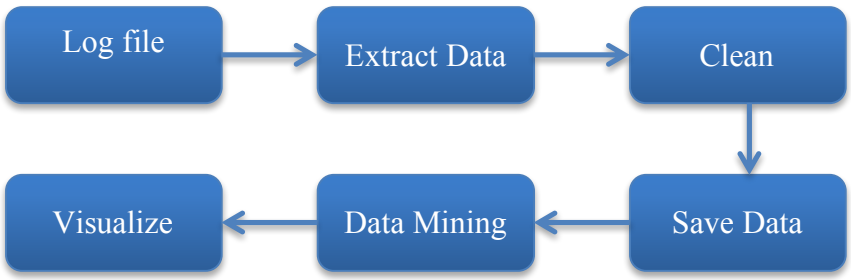


Figure 5-1 Log Analyses

5.2 Implementation

Nagios was the only monitoring solution that was logging monitoring files. It created a dataset that was about 3 months in duration and around 500 MB in size. This would help in getting an insight about the status of the machines and help in understanding the overall system. Python [21] was used to create extract data from the files that were stored on the monitoring server. As there was not a direct way to access the logged files, ssh tunneling [22] was used to retrieve the files on our test node. The figure 5-2 shows a sample log file data.

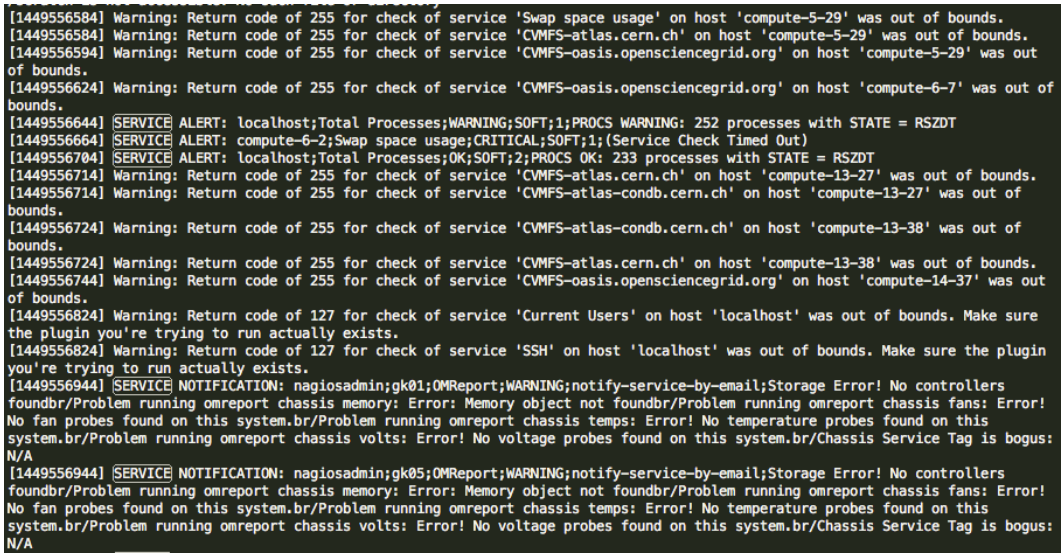


Figure 5-2 Sample log file

After the extraction, the text files were processed to get some quality information about the nodes, which was then sent for cleaning. During the cleaning phase the nodes were labeled based on the type of error triggered by an alert. We categorized the alerts mainly into three categories as service alert, service notification and warnings. The Unknown alerts were filtered out of the system. The alerts were extracted to understand the impact and to derive results. We used WEKA [23] [24] to perform initial analysis to segment data into attributes and quantify results to understand the impact of each. The alerts were then studied using sentiment analysis in order to quantify results. The approach was not able to answer many questions: Does the extracted alerts have a significant impact on the overall cluster health? Can data from only one source derive useful results?

5.3 Experiments with Data

In order to get more insight about the data, we used WEKA [23] [24], a tool used to leverage the use of machine learning algorithms for data mining tasks with the help of a UI (User Interface). In order to use WEKA, we used the extracted logs and converted into a

comma separated file (CSV) using Python scripts. The parsed logs are then fed to WEKA, which is used to analyze the data set. The data captures each machine id and associated attributes for each such as ‘Status’, ‘Error Message’ for machine id ‘compute-5-13’. This representation doesn’t provide any information regarding a broader view of the datacenter. We devise the idea to incorporate machine groups associated with each ID to get a better view into the health of datacenter. After parsing the response, another attribute is added to the data with the group information derived from the name of the entity. We do get a better view into the breakdown of data as seen in the figure 5-3.

Name: MachineGroup
Missing: 0 (0%) Distinct: 16 Type: Nominal
Unique: 0 (0%)

No.	Label	Count
1	localhost	966
2	compute	56362
3	6248.0	4438
4	storage	7373
5	gk03	481
6	gk06	501
7	gk02	558
8	xrdb	937
9	nas	468
10	gk04	494

Figure 5-3 Numerical view of data in WEKA

The figure gives number of alerts generated by a host on a single day.

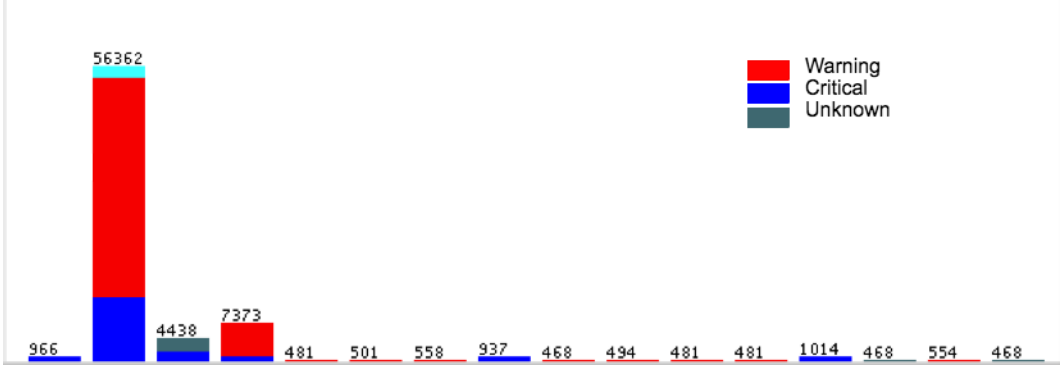


Figure 5-4 Error Messages distribution

The use of WEKA was good for initial understanding but we were not able to perform different queries based on the dataset. The use of Tableau [25] provided a more robust interface to query and visualize data. A similar analysis was done on tableau for finding relation among the machine groups and type of error messages generated. This helped in explaining the high volume of warning reported by compute nodes, which could be used in further analysis by segmenting into different type of machines, figure 5-5 gives a detailed view of errors being generated by different set of machines.

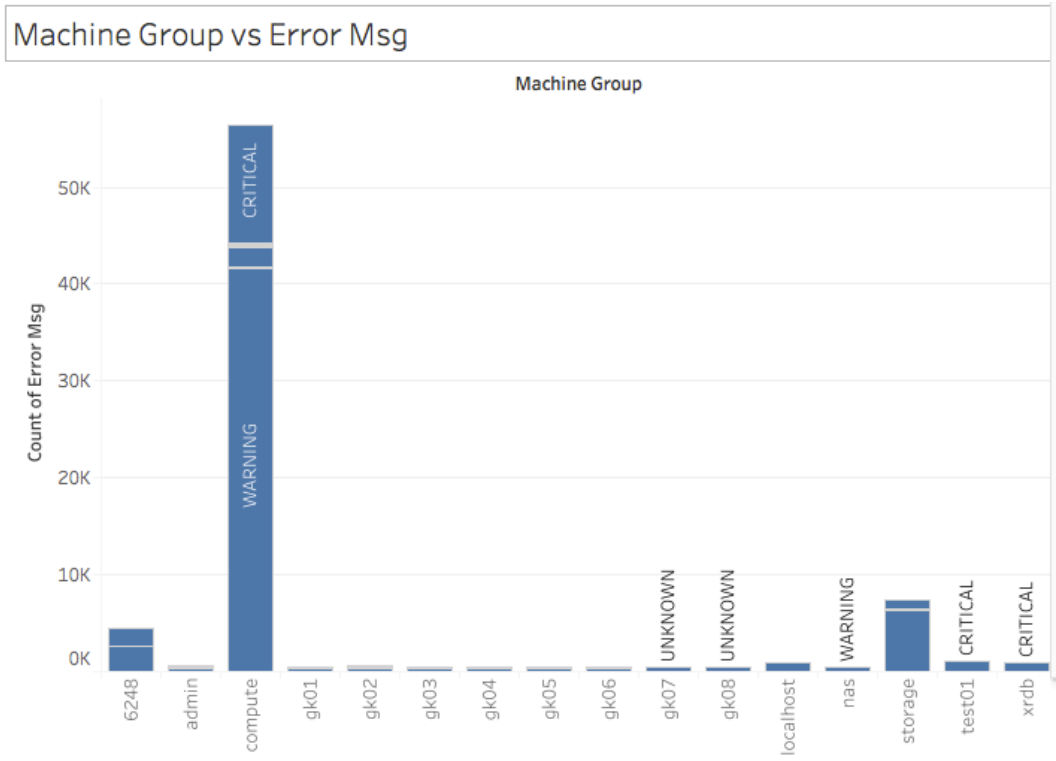


Figure 5-5 Query with Tableau

Considering the high volume of compute nodes generating warnings and error, further analysis was performed to dig more into it. The critical alert generated by compute nodes is filtered, and then the error message is used to plot against the number of alerts. Analysis is performed to

find the errors that have the highest percentage in terms of being logged. Figure 5-6 gives an in-depth into the different type of errors, which are triggered by system.

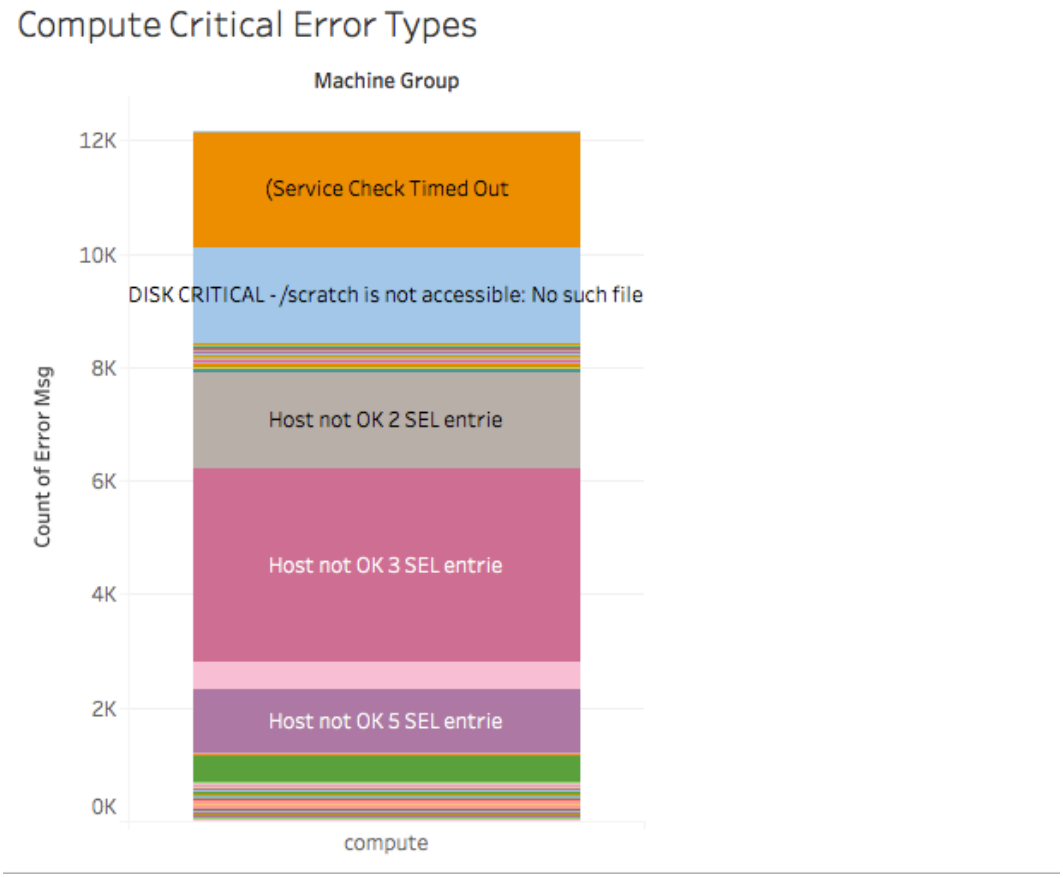


Figure 5-6 Compute Nodes Analysis

5.4 Flaws Found

The idea to extract data from all sources didn't work. The Ganglia system has an internal process of storing and logging data in form of RRD (round-robin database) [26]. The extraction of data over frequent intervals was a challenging task. Secondly, the data monitored by Torque was never stored in any flat files. So, it was not possible to get a holistic view regarding the health of datacenter. The use of WEKA and Tableau did give us a picture to find the most prominent part of the data center "compute nodes", which were returning highest

number of critical errors. However, use of plain visualization was not able to perform an overall health check of nodes in the datacenter. The approach to concentrate on a particular part of the data center would partly solve the problems but cannot be used as a long-term solution to manage the health of the datacenter.

5.5 Need for Improvements

As there was no true source, the need for a new data source was eminent. A new data storage solution was implemented, as we needed data over a long period where we can maintain the state of the data across disparate monitoring systems. The usage of file or database as the source was analyzed in terms of efficiency, maintenance and ease of usage. We choose a database since storing flat files presented us with problem such as time spend on doing IO (Input Output) operations, also maintaining the state of particular machines was much easy in a database.

Chapter 6

Framework

6.1 Introduction

In this section, the methodology and architecture of the framework are described in detail. We have divided the framework into functional layers, each forming a pipeline of processes. The first section will cover the over-all system design with an overview of each layer. The second section will talk about the details of each layer and its components. The third section will describe the implementation of logic for each layer.

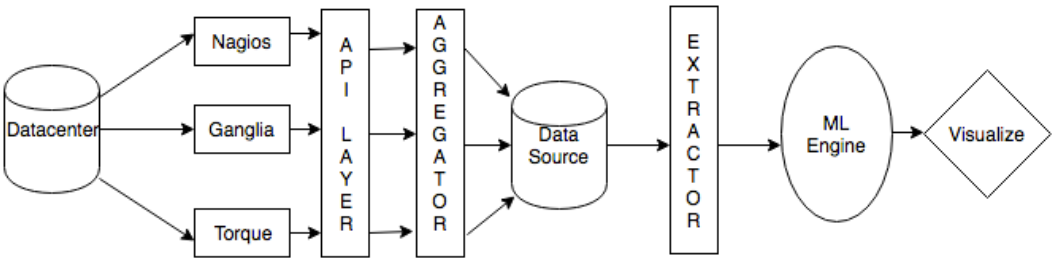


Figure 6-1 Framework Architecture

Figure 6-1 describes the layered structure of the framework. All the 3 monitoring solutions are running independently and a pipeline is created to retrieve the data from disparate systems. To fetch data, an abstract API layer is created on top of monitoring solutions to get results on each API call. In order to get data a second layer is created which is termed as Aggregator, which collects data by making concurrent API calls to the API layer and combine the values based on node names (server names), thus making it a high dimensional data (37 dimensions). The aggregator layers not only combines the data but also helps in cleaning and formulating data to a unified format. The well-formatted data is then pushed to the next stage, where it is stored in a MongoDB database as key and value pair based on the node names and

timestamp. The database acts a single source of truth; it keeps all the values with timestamp and node name, to be retrievable at any time. The next layer is the Extractor, which fetches the stored information on the database based on the timestamp and nodes. The extractor also performs the normalization of the data and sends it further in the pipeline. The data is fed to machine learning engine, where a series of data mining principals and algorithms are applied on the data. The results are then send to the Visualization engine, which renders the final view of data. The end user can change attributes and re-run the ML logic over selected attributes.
A second view based on each node is also created for the user, where a user can see the historic view of a single node based on parameters. To process the visualization for node view data is pushed into a new data source, which is queried for the state of the required node in the last hour.

6.2 API Layer:

In order to provide a generic layer for fetching data we created an abstraction on top of all the monitoring solutions. Each monitoring solution has a different abstraction as in Nagios setup, a ssh pipeline was created to the master node and the results were fetched back to the script. In case of Ganglia, we open a socket to the GMOND Host and fetch the latest status of all nodes reported by Gmond across datacenter. Torque information is fetched using PBS [27], using a library in Python. All the modules are then combined under API to be able to retrieve status of elements on each call.

6.3 Aggregator

This layer performs concatenation of data from all the 3 monitoring solution API calls. It also performs the basic normalization of the data. The values reported by the three monitoring solutions are different and all of them use a different scale to report similar metrics. Memory consumption such as Swap Space is measured in Byte or KB (Kilo Byte) are converted to

MB(Mega Byte). It also performs the cleaning of data to make the data in a key value format. Node name is used as the key and the attributes from all monitoring resources are used list of values.

6.4 Data Source

In order to decide between a SQL (Structured Query Language) and a NoSQL [28] (Non SQL) database, we evaluated both in order to pick the best suitable for our model. The traditional Structured Query Language is beneficial for the use case, which require the implementation of ACID (Atomicity, Consistency, Integrity and Durability) properties. As per the requirement we needed to store data at frequent intervals and then extract the data to implement machine-learning algorithms to do analysis. The database we required should have a high read and significantly low writes performance, which is best supported by a NoSql database.

Secondly, in case of a SQL database, a pre-defined schema is required that remains consistent. If we need to change the schema, the whole database is required to be created again. As per our requirements, which changed frequently, we were unsure about the total list of attributes we may need to cover for creating the schema. NoSql provided us the ease of changing the structure of database anytime without any re-work and therefore saving a lot of time and effort. So the choice of a NoSql database was optimum.

NoSql has many different types of databases based on the structure, we choose mongodb [29], considering the document-oriented database that could store the snapshot of state for all machines in a single json document. The data from aggregator is pushed to a mongodb database, which stores data in key value pair. Along with key value data, we also capture the timestamp of the request and store in the data source.

6.5 Extractor

The extractor is used to fetch data from the data sources. After extraction, the data is again meshed and only the required attributes are fetched from the database. The data goes through second round of cleaning and normalization. The numeric values reported for different attributes are measured on different scales. In order to avoid higher range values affecting the decision model we need to re-scale them. We use in-built function in Python to perform scaling. The fetched data is then pushed to the machine-learning pipeline.

6.6 Machine Learning

In order to choose machine-learning models, first we analyzed the type of model that would fit our scenario. The use of supervised verses unsupervised learning was studied. The node level monitoring data we store in the database by combining the data extracted from all the monitoring solutions was highly dimensional. In order to use the data, which was not labeled, the use of unsupervised learning model was required. Clustering was used to identify different nodes with similarities and the nodes were found, which were very different from the other nodes and could be classified as anomalous. In order to visualize the clustered results, we reduced the high dimensional data, 37 dimensions in our case are reduced to 2-dimensions without losing most important information. Principal Component Analysis (PCA) [30] was performed, which we will be discussing below in detail, in order to have visualization. Figure 6-2 highlights the clustered view of the datacenter. The red nodes seemed to be performing a lot different than the other groups. The blue, green and yellow clusters also showed some differences after clustering in the visualization. After further analysis on the clustered machine types, we found that the clustering was also taking in account hardware aspects of different nodes, which was a desired result in finding similarities and dissimilarities among group of machines.

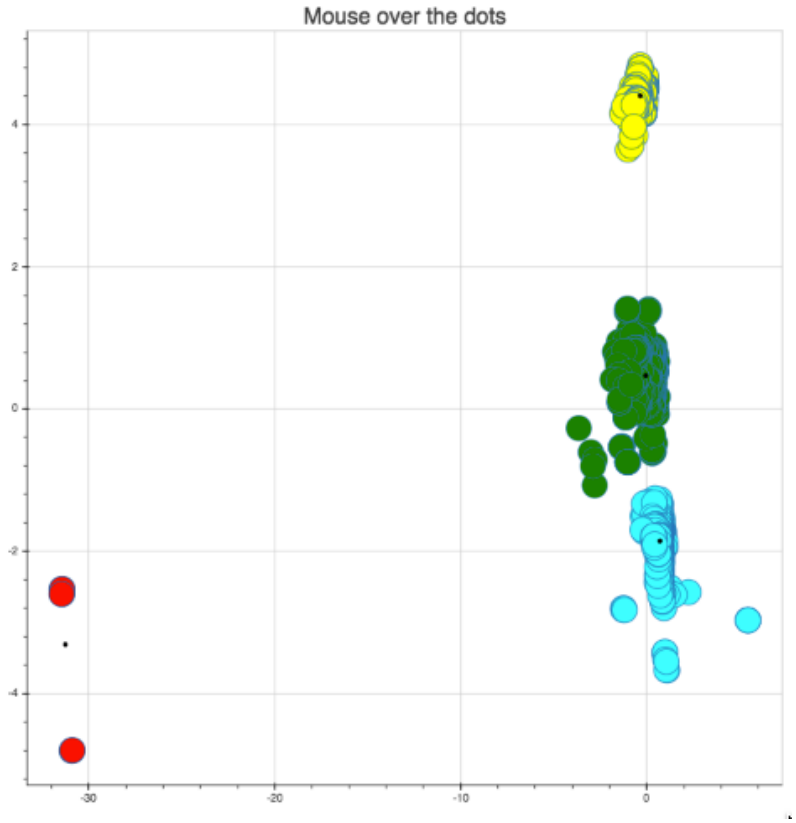


Figure 6-2 Clustered view of the datacenter

In the cluster we had different hardware specification for set of machines as some were older, having 2 Gigabytes of Random Access Memory (RAM), and other were the newer additions to the stack of machines, having 16 Gigabytes of RAM with 4 cores. The clustering results were able to showcase the differences among the machines based on hardware specification rather than being focused primarily on the runtime changing attributes, which could potentially have an impact on the health of nodes in the datacenter. However, the data could find usage in terms of finding the differences in performance of older machine and newer machines. We tried to monitor the performance of both set of machines and tried to understand the patterns. After the analysis it was found out that the machines that were old, were being over utilized whereas the newer set was being underutilized. It was also found out that the problem was associated with the Torque configuration, the monitoring solution used to schedule

jobs. The jobs were based on the amount of RAM available on a node, minimum configuration required to run a single job was 2 Gigabytes of RAM. The older machines had one slot and the newer machines had 8 slots available. The configuration was such that, the machines with 1 slot were always assigned jobs first, even if there were more slots open on the newer machines. The suggested improvement will increase the overall cluster performance as it would be able to process more jobs and also utilization will be balanced across cluster. The efficiency of datacenter will improve and it will become more cost effective.

6.6.1 Principal Component Analysis

The data extracted from all different monitoring solution was a very high dimensional data set having 37 dimensions. As it would be very difficult to make sense out of such high dimensional data, a statistical procedure called Principal Component Analysis (PCA) [31] was used. The 37 dimensions are reduced to fewer dimensions by using the orthogonal transformation for dimensionality reduction. In an iterative manner, at each step a principal component is chosen which has the highest variance (that is, larger variability in data) under the constraint that it is orthogonal to the previous components. Thus it minimizes the possible information loss in the dimensionality reduction. We use Python library scikit [32] to perform PCA on the dataset.

6.7 Measure Performance

As per the previous experiment, the clustering results were mostly based on the hardware specifications and less on the runtime attributes, which could have a significant impact on the overall health of datacenter. In order to understand and measure the system's health we devise new strategies as described in the following section.

6.7.1 Normalize by Total Slots

The datacenter was comprised of different type of machines based on the hardware specifications. Each node comprised of some number of slots based on the RAM (Random Access Memory) available on board and a single slot consumed 2 Gigabytes of RAM. The idea was to take into consideration usage data across cluster and then we could optimize the data by normalizing with Total Slots. Total slots were defined as the number of job slots available on each server and each job allocation could consume single or multiple slots based on the request. We normalized all the monitoring data attributes that are reported by the monitoring solution by the total number of slots on board. This divided the usage of all the datacenter resources into per slot metrics. After the initial run, we learnt that we are still facing the same issue. The total slots did normalize the datacenter resources but didn't take into account their usage at runtime.

6.7.2 Normalize by Committed Slots

A technique was devised in order to suppress the hardware aspects within the dataset by introducing a new term "Committed Slots", which signified the number of job slots occupied on each server. As per the design, based on the job type (single core or multi core) 1 or 8 cores were assigned across the cluster. Depending on the committed slots we normalized the resources metrics (Swap used, Load) to have more usage centric data. The attributes that reported static data were eliminated as they were related to hardware specifications such as the total memory, total RAM etc. We were left with usage of a resource by per committed slot. These attributes in combination would give a better understanding of the overall health of the datacenter. Figure 6-3 gives the view of the cluster's health normalized by committed slots.

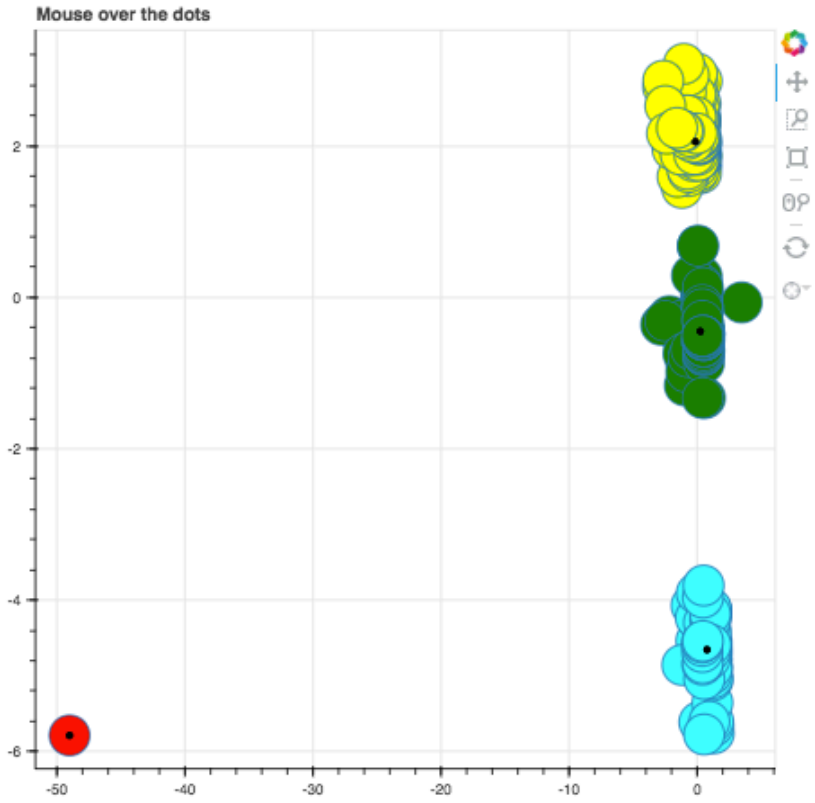


Figure 6-3 Snapshot by commited slots

6.8 Choosing Attributes

After analyzing the previous results, it was found that the hardware specifications were still hindering the clear health snapshot because attributes like ‘Total slots’ available were still indirectly relating to the static machine configuration. In order to better understand and classify data, we choose attributes from different monitoring solutions and studied the importance of each as described below:

6.8.1 Nagios

The main responsibility of Nagios is to report the state of services running on each node along with memory related metrics. In order to choose among the attributes we choose the ones reporting state of different services on each server. Table 6-1 gives a detailed description of the attributes and the description.

Attributes	**Description**	**Value**
Swap_State	Status of swap space used	0 - 3
IPMI_State	Intelligent Platform Management Interface (IPMI) status in machine	0 - 3
IPMI_Info	Detailed Information, gives quantative information which is extracted about host	Detailed information based on current state
FreeSpace_State	Status of free space remaining	0 - 3
CVMFS-OSG_State	Service status for Open Science Grid(OSG)	0 - 3
CVMFS-CERN_State	Service status for CERN	0 - 3
CVMFS-CONDB_State	Service status for CONDB database	0 - 3

Table 6-1 Nagios Attribute description

Code	**Status**
0	OK
1	Warning
2	Alert
3	Unknown

Table 6-2 Status of attribute values

6.8.2 Ganglia

It reports metrics related to almost all type of resources being used on a server. It covers all the metrics as mentioned in table 6-3.

Metrics Areas
CPU
Disk
Load
Memory
Network
Process
System

Table 6-3 Metrics in ganglia

We focus primary on attributes, which were directly related in the job running condition such as CPU_WIO (Wall time spent while waiting for Input output operation), load on the machines, memory cached. Other attributes were not extracted from the system.

6.8.3 Torque

It is responsible for scheduling jobs received from different sources. It assigns jobs to different queues depending upon the type of job and requested load. All the jobs are run on compute nodes and each compute node is monitored, as it would send a heartbeat every second to the torque system. In case the heartbeat is not received the node is marked as ‘Down’. Once the node is marked, as ‘Down’, no jobs would be run on that node. Torque metrics overlap some of the reported values with that of Ganglia. We don’t eliminate similar values to get a secondary view of attributes. We use the attributes as mentioned in table 6-4.

Attributes	Description
T_State	State of current node
T_Slots	Total slots
T_SlotsUsed	Total slots used
T_AvailMem(MB)	Available RAM
T_Time_Last_Rec	Last heartbeat time stamp
T_LoadAve	Load average

Table 6-4 Torque attributes

6.9 Attributes of Importance

We choose all the important attributes and store them in database for analysis. After performing PCA [31] and clustering we were able to find the anomalous nodes. We also wanted to understand the importance of attributes, which make these nodes anomalous. It would also us to help in better management of health of datacenter by resolving issues related to particular attributes.

6.9.1 Feature extraction

We use feature extraction to find out important features from the high dimensional data set. We use forest of trees to evaluate the importance of features. The high variant features, which provide more insight into the dataset, are significantly more important than the others. The figure 6-4 shows the descending order of attributes.

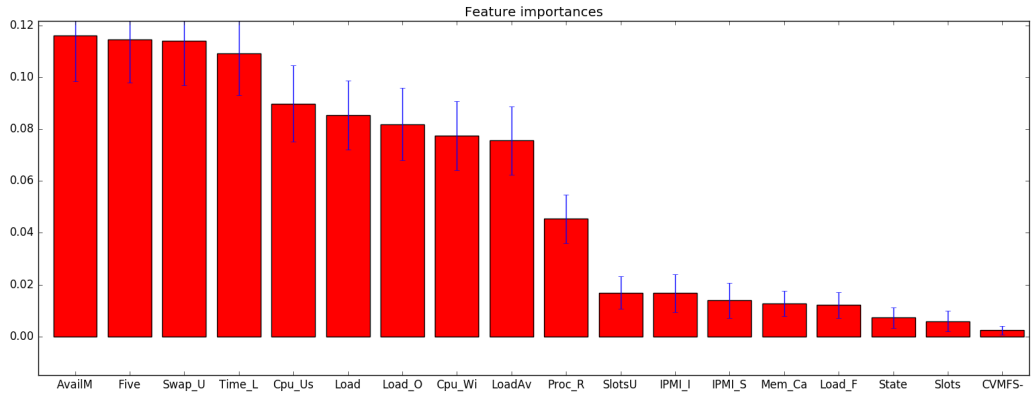


Figure 6-4 Feature Importance

6.10 Visualization

We start the visualization using the Bokeh [33] library that can be configured to display the data along with tooltip that would exactly point out the machine names for each point in the new dimensional space. We are able to see the clustered data points after performing k-means clustering. Figure 6.4 shows the exact clusters.

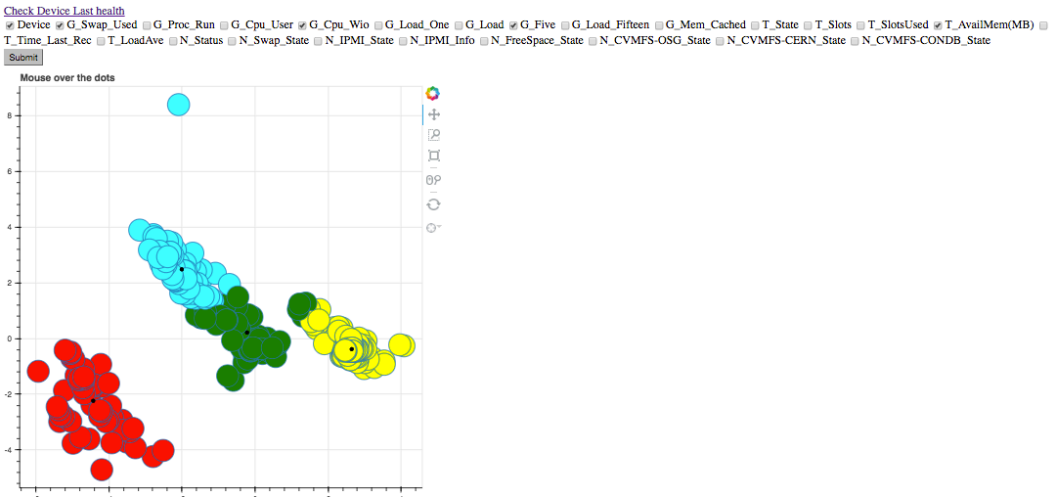


Figure 6-5 Visualization

6.11 Store the device state

While visualizing the 2 dimensional view of the datacenter, we wanted to keep the datacenter information based on the health of each node. The feature to be able to go back in

time for a particular device would help in better triaging the issues that had already occurred. Each run stores the device health information with time stamp in the mongodb database. We created a different API call to render the results from data source. Figure 6-6 shows the dashboard that could be used to monitor each device back in time.

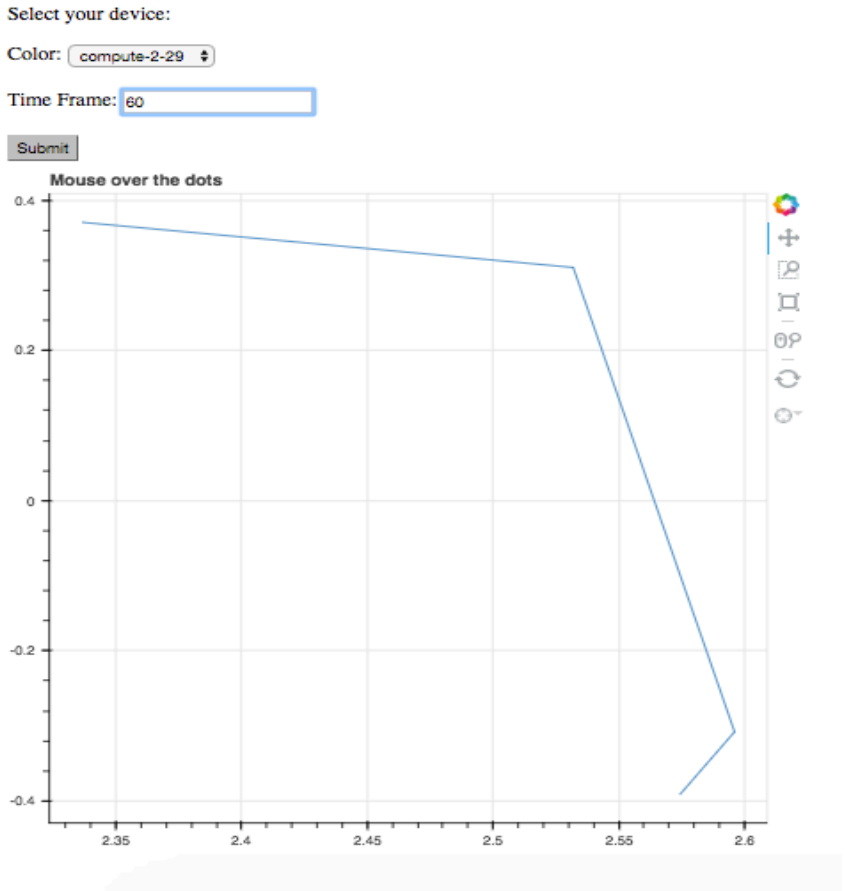


Figure 6-6 Device State

Chapter 7

Summary and Conclusion

This research provides a monitoring solution for a datacenter by identifying anomalous devices by using streaming data generated by different monitoring solution for all the nodes inside a datacenter. Dealing with inflow of disparate data from monitoring solutions was a challenging task. The data required cleaning and scaling to make it suitable for any type of analysis. We start with a log monitoring approach, which is negated due to factors discussed in previous chapters. The focus is then shifted to a layered system with machine learning based monitoring framework. The approach focuses on the usability of existing infrastructure with a plug and play framework, which could be used with different monitoring solutions.

Instead of having different monitoring solution covering all the aspects of datacenter, the idea of having a single high dimensional view of the health of datacenter is presented. The setup is prepared to abstract the monitoring data inflow from given monitoring solution like Nagios, Ganglia and Torque. The layered solution is implemented by having a clear segmentation of task at each stage of the process. As the design includes a clear data source, the state information of machine is further used to derive interesting facts, which remained hidden in the older monitoring process. Machine learning framework adds the analytical ability in the process of monitoring health of the cluster. It enables the administrator to be able to run different algorithms that can be used to draw correlation among disparate attributes that was impossible before.

An interactive user interface is designed to facilitate a faster and intuitive process for analyzing the health of datacenter. It incorporates visualization, which gives a snapshot of the current state of the datacenter at glance after running of pipeline of machine learning algorithms. The interface gives users the ability to select or deselect different attributes they

wish to run against the pipeline. It also includes the ability to track past performance of machines by visualizing the historical health of individual machines.

Monitoring of datacenter is an important problem and even a small improvement in datacenter availability can save significant resource spending. This thesis presents the idea of using a single high dimensional view from disparate monitoring solutions and designing the machine learning based framework on top.

Chapter 8

Future Work

As datacenters around the globe are growing rapidly, the Bunny approach will be able to significantly improve the efficiency. Even by having a small impact on the datacenters, it will be able to save thousands of dollars. Being a layered system, Bunny could be easily integrated with other monitoring solution. It could also be extended to use other machine learning algorithms, which further may provide a better understanding of the anomalous devices and insightful results. It could be used to detect hidden correlation among devices and their attributes.

In this study, the approach discussed involves higher number of read operation. The introduction of cached data storage will improve the performance of the system. In order to make a generic solution, we could add a GUI (Graphical User Interface) based layer, which would provide an interface to users to upload the dataset and perform initial cleaning with selection of data attributes.

In order to make highly scalable solution, micro-services architectures could be used in designing the application. In micro-services based architecture, each functionality can act as an independent service. For example, data extraction and cleaning, machine learning, visualization can all act as independent services.

We could use the temperature cooling system data in conjunction with the data from cluster performance monitoring system. This correlated data could be used to find useful patterns in the system, which can be eventually used to reduce the operational cost involved in running a datacenter. As cooling systems consume most of the power in datacenter, we could save substantial of our resources.

References

[1] CERN. (1954, September) ATLAS. [Online]. https://home.cern/about/experiments/atlas

[2] Nagios. (1999, March) Nagios. [Online]. https://www.nagios.org/

[3] Ganglia. (2001, January) Ganglia. [Online]. http://ganglia.info/

[4] Torque. (2003, January) Adaptive Computing. [Online]. http://www.adaptivecomputing.com/products/open-source/torque/

[5] Dirk deRoos, Krishnan Parasuraman, Thomas Deutsch, James Giles, David Corrigan Paul Zikopoulos, *Bringing big data to the enterprise* , 1st ed., Roman B. Melnyk, Ed. San Fransico, CA, US: Mc Graw Hill, 2013. [Online]. https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[6] D. Van der Wallen, L. Lewis A. Knobbe, "Experiments with data mining in enterprise management," in *the Sixth IFIP/IEEE International Symposium on Integrated Network Management, Distributed Management for the Networked Millennium.*, Boston, 1999, p. 51.

[8] Panita Pongpaibool, Sophon Mongkolluksame, Koonlachat Meesublak Chavee Issariyapat, "Using Nagios as a Groundwork for Developing a Better Network Monitoring System," in *Technology Management for Emerging Technologies (PICMET), 2012 Proceedings of PICMET '12*, Vancouver, 2012, p. 533.

[7] Kenneth P. Birman,Werner Vogels Robbert Van Renesse, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 164-206, May 2003.

[9] I. Rish , A. J. Oliner , M. Gupta , J. E. Moreira , S. Ma , R. Vilalta , A. Sivasubramaniam R. K. Sahoo, "Autonomic computing features for large-scale server management and control," in *In AIAC Workshop*, Rhodes Island, 2003.

[10] Wikipedia. ATLAS experiment. [Online]. https://en.wikipedia.org/wiki/ATLAS_experiment

[11] CERN. Large Hadron Collider. [Online]. http://home.cern/topics/large-hadron-collider

[12] CERN. (1954, September) CERN about us. [Online]. http://home.cern/about/computing

[13] Wikipedia. Standard Model. [Online]. https://en.wikipedia.org/wiki/Standard_Model

[14] CERN. What is atlas. [Online]. http://atlas.ch/what_is_atlas.html

[15] CERN. Data prcocessed at CERN. [Online]. https://home.cern/about/computing/worldwide-lhc-computing-grid

[16] Wikipedia. XRDB. [Online]. https://en.wikipedia.org/wiki/Xrdb

[18] Garrick Staples, "TORQUE resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, 2006.

[17] Digital Ocean. (2014, August) Digital Ocean. [Online]. https://www.digitalocean.com/community/tutorials/introduction-to-ganglia-on-ubuntu-14-04

[19] Wikipedia. (2003) Torque. [Online]. https://en.wikipedia.org/wiki/TORQUE

[20] Wikipedia. Portable Batch System. [Online]. https://en.wikipedia.org/wiki/Portable_Batch_System

[21] Python Software Foundation. (1991, Februrary) Python. [Online]. https://www.python.org/

[22] Ubuntu. SSH Tunneling. [Online]. https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding

[23] University of Waikato. (1993) WEKA. [Online]. http://www.cs.waikato.ac.nz/ml/weka/

[24] Wikipedia. Weka. [Online]. https://en.wikipedia.org/wiki/Weka_(machine_learning)

[25] Tableau Software. (2003) Tableau. [Online]. http://www.tableau.com/

[26] RRDtool. Wikipedia. [Online]. https://en.wikipedia.org/wiki/RRDtool

[28] GNU. (2009) Wikipedia. [Online]. https://en.wikipedia.org/wiki/NoSQL

[27] MRJ Technology Solutions. (1991) Wikipedia. [Online]. https://en.wikipedia.org/wiki/Portable_Batch_System

[29] MongoDB Inc. (2007) MongoDB. [Online]. https://www.mongodb.com/

[30] Kim Esbensen and Paul Geladi Svante Wold, "Principal Component Analysis," in *Chemometrics and Intelligent Laboratory Systems*. Amsterdam, The Netherlands: Elsevier Science Publishers B.V, 1987, pp. 37-52.

[31] Wikipedia. Principal component analysis. [Online]. https://en.wikipedia.org/wiki/Principal_component_analysis

[32] Maintained by OSS. scikit-learn. [Online]. http://scikit-learn.org/

[33] Continuum Analytics. (2015) Bokeh. [Online]. http://bokeh.pydata.org/en/latest/

Biographical Information

Ravneet Singh Sidhu joined the University of Texas at Arlington in fall 2014. He received her B.Tech in Computer Science from Jaypee Institute of Information Technology, Noida in 2011.

He worked as a software developer for Accenture Pvt. Ltd in India for 3 years. In United States he intered with T-Mobile as Cloud Intern and is currently working with Facebook as Automation Specialist.

His research interests are in areas of machine learning, distributed system and software development.