UNIVERSITY OF OSLO
Department of Informatics

Interoperability in Monitoring and Reporting Systems

Nishes Joshi

Network and System Administration

Oslo University

May 23, 2012

Interoperability in Monitoring and Reporting Systems

Nishes Joshi

Network and System Administration
Oslo University

May 23, 2012

Abstract

Monitoring and Reporting are an integral part of IT infrastructure. Lack of single tool that covers all the need for monitoring in an organization has led to uses of different kinds of monitoring tools within an organization. Existence of diverse monitoring tools has led to significant heterogeneity in an infrastructure between the tools and the data they measure.The monitoring tools generally differs in the data structures and interaction interfaces that they implement. One of the key issue in todays monitoring scenario is the interoperability between several monitoring system and the heterogeneous data they produce.

In this thesis some of the current monitoring problems that arises due to diversity in monitoring tools are briefly discussed. It also discusses how a standard common data model can benefit the interoperability between the tools. The implementation of simple proof of concept integration between selected monitoring tools is also explored. The results of implementation indicates the benefits of information interchange between the tools for solving many problems or difficulties that are discussed. The thesis also addresses some challenges in data integration between different tools.

Acknowledgments

Firstly I would like to thank my beloved family, without whose love, support and sacrifices none of this would have been possible.I also want to thank my girlfriend Namrata Pradhan, for her help and support through out the whole process.

I would specially like to thank my supervisor Ismail Hassan for his precious and valuable support, suggestion and constructive critics in regards to this master thesis. I am also grateful for professor Mark Burgess for his crucial guidance, support and ideas for this master thesis.

Lastly I would like to express my sincere thanks and respect towards University of Oslo and Oslo and Akershus University College of Applied Sciences (HiOA) for providing me opportunity to become part of this master program.

Contents

Chapter 1

Introduction

1.1 Monitoring and Reporting

Nowadays our system infrastructures are becoming more and more complex with very high level of dependencies between different components. And having a successful architecture with quick maintenance time , minimum downtime is quite essential when it comes to business critical systems. Our systems are growing complex and large in scale and there is a huge need to monitor those system and keep us informed about the state of the system.This is where monitoring tools comes into picture. Monitoring tools are one of the critical part of the whole architecture of the infrastructure that we build. Monitoring the system and altering when something needs to be watched are the most essential things that system administrators need to perform on daily basis.

The benefits of the monitoring and reporting systems are quite huge and cannot be ignored. Some of the main features or benefits are

Identification of bottlenecks When services are not running according to the desired quality level, monitoring can help trouble shoot the cause quickly. A slow system may be due to various reasons such as slow cpu , slow I/O access , low memory , congested network etc. If a good monitoring systems are in place these situation can be quickly identified and resolved.

Capacity planning In a perfect world, administrators prepare in advance in order to avoid performance bottlenecks altogether, using capacity planning tools to predict in advance how servers should be configured to adequately handle future workloads. Monitoring networks provides very valuable information that can help in making decision. Through data gathering and analysis of network performances , history of previous data trends , high level view of the architecture , one can answer about the performance and capacity of the infrastructure and make informed decision when needed. The goal of capacity planning is to provide satisfactory service levels to users in a cost effective manner.

Security Monitoring is also applicable in security area. It can automate system interrogation and can easily detect anomaly.It can set sensors on high risk systems and can improve intrusion detection and incident response and can greatly improve forensics.

Chapter 2

Background and literature

2.1 What is monitoring and Reporting

Basically in simple terms Monitoring involves periodically taking measurements and checking some system state or variable to present or detect correctness of the state or service that is being monitored. Some example of monitoring would be to periodically checking the machine if its running or not or checking some service state if its actually functioning as required. While the task seems to look easy for small system to do it manually , it becomes hugely complex and even impossible due to sheer number of monitoring that has to be done on a huge system. That is why there are lots of monitoring tools available today to solve these problems with different solutions. Monitoring tools are critical to quickly identify and address problems that affect an IT organizations service. These tools scan for problems with network, systems, and application resources that a company relies on for its business. In general goals for these monitoring softwares can be briefly summarized as below

- monitor the system if its working as required by the specification, business goals
- monitor the system for its QoS(Quality of Service), ensure the system is performing efficiently as expected.
- detect anomalies by comparing its previous state or known state
- provide real time information about the monitored system

On the other hand, Reporting goes hand-in-hand with Monitoring. Reporting term is often referred as to present the monitored values to give it a meaningful context generally by presenting it with some short summaries from some statical analysis or displaying it in meaningful visualization such as graph, charts so that the it can be analyzed and interpreted quickly to detect the state of monitored system.

Previously getting the data from the system was considered a problem , but now collection of data has become a easier task , but managing such huge collected data and analysis of those data's for something meaningful has become a challenge.

> In a data deluge–era sensing system, the number and resolution of the sensors grow to the point that the performance bottleneck moves to the

sensor data processing, communication, or storage subsystem[1] .

Todays modern monitoring systems can generate huge amount of data. Our infrastructure , business , services , resources everything is scaling out very quickly and so is the stats or the monitored variable with them. Just storing the generated data's for such system has become a challenge let alone getting something comprehensible meaning out of them. But as our business requirement highly depends on the IT infrastructure, we cannot ignore the monitoring and reporting part but must be able to quickly analyze , predict and be able to take action whenever necessary.

As this need has identified by many , there are numerous solution out there to handle the situation. There are lots of tools out there to solve or tackle the mentioned problem. As tools has emerged they have taken very diverging solution to achieve the goals , some tools specializing in some area like collecting data , monitoring while some tools excelling in presenting the data's.

As we know there is no perfect solution or "one tool to rule them all", We are inclined to piece together many different tools to achieve our business goals. Many tools in these area can be categorized in different parts [2] mainly

- Collect
- Transport
- Process
- Store
- Present

In current scenario tools for monitoring or reporting is either one of the part or some mixture of the part. Some solution does the collection part quite well leaving off other areas while other tool fits the bill in other areas. Some examples are Collectd [1]. "collectd is a daemon which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways, for example in RRD files." [1] As good as Collectd is in collecting performance data it doesn't provide presentation of the data in graphical way like graphs or charts. GroundWorks [2], Graphite [3] are some tools that has been ahead in visualization and presentation part.

2.2 Software comparison

As there are so many tools to accomplish the same task, choosing or comparing them is often not easy or straightforward. Since the quality of products varies widely, both industry and the research community have reported several evaluation methods that are tailored to the specific characteristics of product. However, research has shown that practitioners rarely use formal selection procedures [4].

Probably the most typical problem in software evaluation is the selection of one among many software products for the accomplishment of a specific task. software

[1] Dr. Richard G. Baraniuk Science Magazine 02/2011

[2] http://serialized.net/2011/02/getting-more-signal-from-your-noise/

evaluation may have different points of view and may concern various parts of the software itself, its production process and its maintenance. Thus, software evaluation is not a simple technical activity, aiming to define an ”objectively good software product”, but a decision process where subjectivity and uncertainty are present without any possibility of arbitrary reduction. [5]

Basically evaluating or comparing the software contains evaluating certain important attributes of the software which includes

- functionality
- cost
- market share
- support
- maintenance
- reliability
- performance
- scalability
- usability
- security
- flexibility
- interoperability
- legal/license issues

Some of the previously invented evaluation framework which are mostly relevant for OSS (open source softwares) are

- Comparative assessment of open source software using easy accessible data [6]
- Method for Qualification and Selection for open source software [7]
- A Quality Model for Open Source Software Selection [8]

2.3 Monitoring terminologies

Instead of a single, centrally managed system, systems have become widely distributed, with the importance and sophistication of each component increasing. Issues such as availability, performance, fault identification and diagnosis all became greater challenges to system administrators. In the extreme case, the exponential growth of the Internet presented a clear need for standard management approaches. Following some early protocols such as the Simple Gateway Monitoring Protocol (SGMP), the Simple Network Management Protocol (SNMP) gained early acceptance as the management protocol of choice for IP networks.

2.3.1 SNMP

The Simple Network Management Protocol (SNMP) has been the most widely used method for network management on the Internet since it was introduced in the late 1980s. However, SNMP has been used mostly in monitoring for fault and performance management,but was hardly used for configuration management due to its limitations [9].

SNMP (Simple Network Management Protocol) is the common language of network monitoring. It is integrated into most network infrastructure devices today, and many network management tools include the ability to pull and receive SNMP information. SNMP extends network visibility into network-attached devices by providing data collection services useful to any administrator. These devices include switches and routers as well as servers and printers. The structure of SNMP is governed by Management Information Bases (MIBs)[10] which describe what each data entry represents.

A MIB is a database used to store management information in networks. They are used by network management systems to identify network data objects that are stored, retrieved and set by the system. A MIB uses a hierarchical tree structure to store an extensible collection of data. A subset of Abstract Syntax Notation is used to specify an object. The MIB used for SNMP is standardized, however, it allows private organizations to insert custom objects into the structure. An individual organization can extend the standard MIB defined by SNMP by adding to the tree below an assigned node under iso.org.dod.internet.private.enterprise. Cisco for example has been assigned number 9 under enterprises. So, Cisco creates MIBs entries for its own devices under iso.org.dod.internet.private.enterprises.cisco.

Modern computer networks require that thousands of different devices from hundreds of different manufacturers successfully communicate with each other. For this reason, standards have been developed to ensure that products from different manufacturers inter operate successfully. In the network monitoring world, standards such as MIB-II have been designed to provide an easy way for network management products to inter operate with the large amount of devices on the market. Unfortunately, technology often evolves faster than the standards that are developed to carter them. So keeping both of them in sync is very difficult to achieve.

The SNMP MIB-II tries to tackle this problem by providing the enterprises object identifier, where manufacturers can create their own, device specific, MIB trees. As technology advances, the gap between the standardized MIBs and the requirements of the technology increases. This results in more and more information about a device being stored under the enterprises OID, to the point where, when dealing with modern technologies, a vast majority of the management information about the device is only available through the enterprises OID.

Network management tools using these enterprises MIBs are, in general, limited to working with devices from one vendor. The effect of this limitation is that, in order to fully manage a network using these tools, organizations are required to implement a homogeneous network that is, a network where all network infrastructure and monitoring tools are sourced from a single vendor.

Unfortunately this is not always desirable or possible. Networks tend to grow in an ad-hoc fashion, depending on the demands of the users of that network. This is

especially true of small organizations that do not have the resources to implement long term strategic plans. The result is usually a heterogeneous collection of devices from different manufacturers.

Although widely used, SNMP is not without its limitations. SNMP is a polled system. It requires that monitoring stations actively poll devices for information. This can result in significantly increased network traffic if the polling frequency is not carefully monitored. That is, as the network expands, the amount of management data to be processed and transferred between managers and agents is continuously increasing. Therefore, SNMP is insufficient to manage these huge and continuously expanding networks because of constraints in both scalability and efficiency. Many other disadvantages are also encountered on the usage of the SNMP for large scale monitoring as mentioned on the paper [11].Some of them are

- Manager station becomes the processing bottleneck as the architecture grows.
- Manager station becomes the single point of failure , which may cause inavailability of the monitoring services totally.
- Since SNMP is UDP based , the message exchanged between manager and agent is limited by the size of a single UDP datagram.
- Sometimes Network distance between Manager and network elements can be factor for reliably controlling them , due to inherent instability imposed by long control loops

2.3.2 CIM

Common Information Model(CIM) provides a common definition of management information for systems, networks, applications and services, and allows for vendor extensions. CIM's common definitions enable vendors to exchange semantically rich management information between systems throughout the network.[12].

One of the goals of the CIM model is to consolidate and extend existing management standards and instrumentation such as Simple Network Management Protocol (SNMP), Desktop Management Interface (DMI), Common Management Information Protocol (CMIP), and so on. It does this by using object-oriented design and constructs. CIM objects include computer systems, devices (like printers and batteries), controllers (for example, PCI and USB controllers), files, software, physical elements (like chassis and connectors), people, organizations, networks, protocol endpoints, policy, and more. CIM also is evolving its eventing object hierarchy and mechanisms. CIM is part of the DMTF's overall Web-based Enterprise Management (WBEM)[13] initiative. WBEM includes CIM as the data definition, XML as the transport/encoding method and HTTP as the access mechanism. CIM looks very promising for future interoperability between different tools which depends upon different tools adaptation, implementation and support for it as well as the demand from the users.

2.3.3 RMON

Remote network monitoring (RMON) [14] is the standard of how to monitor Internet traffic. This is a standard that is supposedly implemented by Internet device vendors

so that a network using RMON-compliant devices can be monitored using RMON-compliant software. The original version (sometimes referred to as RMON1) focused on OSI Layer 1 and Layer 2 information in Ethernet and Token Ring networks. It has been extended by RMON2 which adds support for Network- and Application-layer monitoring and by SMON which adds support for switched networks. It is an industry standard specification that provides much of the functionality offered by proprietary network analyzers. RMON agents are built into many high-end switches and routers.

An RMON implementation typically operates in a client/server model. Monitoring devices (commonly called ”probes” in this context) contain RMON software agents that collect information and analyze packets. These probes act as servers and the Network Management applications that communicate with them act as clients. While both agent configuration and data collection use SNMP, RMON is designed to operate differently than other SNMP-based systems:

- Probes have more responsibility for data collection and processing, which reduces SNMP traffic and the processing load of the clients.
- Information is only transmitted to the management application when required, instead of continuous polling

2.3.4 NETCONF

While SNMP is still in wide use , the IETF considered it outdated. The NETCONF working group was chartered to create a replacement protocol, based on XML. This replacement is the NETCONF Configuration protocol [15]. It is a protocol designed for installing, modifying and deleting the configuration of network devices. The operation carried out by NETCONF are carried on top of a Remote Procedure Call (RPC) layer using an XML encoding.

Similar to how MIBs define the data structures for SNMP, data structure for NETCONF are defined using the Document Schema Definition Language (DSDL). The Netmod working group is chartered to draft the translation mechanism between DSDL schema’s and NETCONF XML.

2.3.5 RRDTool

Nagios , being a framework does not try to persist the data by default but has a different ways and API that lets other system tap into its collected data. [16, Chapter 8]. Many tools utilizes these features and come up with their own storage solution which can be efficiently utilized for analysis. The most popular and solution to store performance and monitoring matrix is Round Robin database.

In a round-robin database (RRD) usually time-series data like network bandwidth, temperatures, CPU load etc. is stored. The data is stored in the way that system storage footprint remains constant over time. This avoids resource expensive purge jobs and reduces complexity. CFEngine [17] , collectd, groundworks [2] all utilize RRD storage scheme. But when extracting and utilizing the data they all represent or model the data differently for internal purpose.

RRD is the defacto standard to store time-based data and is frequently used to store network performance data. RRD files contain a binary format developed by Tobi

Oetiker for his Multi Router Traffic Grapher (MRTG), and is now in use by countless other tools. The rrdtool command line tool allows easy parsing, storing and conversion of the data. Rrdtool can import and export data from and to XML format. The main advantage of the RRD format is its fixed database structure that doesnot grow beyond certain size and as well as the tools that are available to manipulate the data. RRD files only contain little meta data, such a short description what was measured, the duration and interval. It does not provide an ontology for metadata about the measurements to describe exactly what was measured where and how. Such meta data are required if different RRD data source are to be merged or automatic detection of anomalies is to be done.

2.4 Interoperability between tools

Interoperability is defined as ability of the tools to interact between different heterogeneous tools that are currently present or any future tools without needing to change the architecture of tools that are interacting. In other words it can be simply described as ability of two or more component of system to easily interchange data for consumption. Interoperability leads to innovation which results in different system components that works together to increase the efficiency of the overall operation also giving end users lot of freedom in choosing the right tools for the job.

The tools that are currently available have inefficient interoperability between them. Mainly there are two types of interoperability that are mainly identified. They are Syntactic interoperability and Semantic interoperability.

2.4.1 Syntactic interoperability

Syntactic interoperability refers to the system where they are able to exchange data and communicate with other systems. They are usually achieved by the means of specified data format, protocol etc. It is usually tackled in application layer. It is one of the primer for achieving further interoperability between systems.

2.4.2 Semantic interoperability

When the systems overcome the difference between each other at knowledge level then it is referred as Semantic interoperability. It means that the systems will be able to infer knowledge, automatically interpret data, context in meaningful way to provide useful result without any ambiguity or inconsistencies between two systems. Semantic interoperability is therefore concerned not just with the packaging of data (syntax), but the simultaneous transmission of the meaning with the data (semantics). To achieve semantic interoperability, systems must be able to exchange data in such a way that the precise meaning of the data is readily accessible and the data itself can be translated by any system into a form that it understands[18]. A major challenge in achieving semantic interoperability is the lack of explicit and compatible semantics in the digital design representations. These are hard to overcome problems but are being addressed in other domains like knowledge management and artificial intelligence using ontologies[18]. Some projects like CIM [12], Semantic web [19] has tried to tackle this problems.

Chapter 3

Problems

With availability of varied solution and tools for monitoring and reporting, it becomes increasingly difficult to manage the ecosystem and takes huge amount of resource and time just to have the whole monitoring ecosystem in place. And with wide diversion in tools and solution we use , we increase the cost as well as complexity of the solution. Some of the problems these induced heterogeneity in system are described below.

3.1 Common interface

We may never achieve a tool that solves all our problems at once and its better if it does not try to do that, else the tool will just be monolithic and too complex. Instead we should adapt for modularization and build tools confirming to certain standards whose solution can be reused and easily adapted to used by other tools.

Most existing monitoring tools collect resource information and try to model them to closely integrate with their own conceptual or GUI module.The disadvantage of this is that other subsystems or applications that may need the same data are not able to easily utilize the monitoring informations provided by other tools.

Example we would have many different monitoring tools in our system due to technical reasons , historical decision etc. They will be using different information models to store or process their data. Some of the well known tools among many monitoring solutions are CFEngine [17], Nagios [20], GroundWorks [2], Cacti, Ganglia [21] etc. They all have different application interfaces and data model. They have different data storage model adopted to their own requirements. Exchanging information or getting information out of these tools requires lots of effort like writing wrappers or custom solutions interacting with their API's etc.

If all the tools had supported standard representation of data model then the task of exchanging information between tools would have been very efficient. If there was some common pattern or standard , then other tools in ecosystem would easily tap into the data to provide extra functionality not provided by the native tools.

3.2 Scalability

Our Infrastructure are very dynamic in nature. We are never for sure how much expansion are we going to have in our system. So the tools we choose for our infrastructure

should be flexible enough to handle the current scale and also the future. The monitoring tool should scale in capacity to manage a cluster composed of a large number of computers, without consuming lots of system resources. It should provide the largest degree of parallelism to facilitate the concurrent control and monitoring of the underlying cluster resources. The monitoring tool should exploit the parallelism of the execution of monitoring task, providing group control and inspection, and reducing overheads in coordinating monitoring agents and server. [22]

One of the key challenges faced by distributed systems is scalable monitoring of system state. Given a large enough collection of nodes and the associated computational, I/O, and network demands placed on them by applications, failures in large-scale systems become commonplace.[23]. In large scale environment, interaction between different subsystems and resources can lead upto very complex interaction. They can generate huge amount of data which may tend to generate large network traffic and resource consumption. Monitoring and reporting tools need to be capable of effectively handling those situations for analyzing and diagnosing any problems quickly and effectively. Having a single point of reference to monitor huge distributed system would be a key factor to operate efficiently in large heterogeneous environment.

3.3 Data Visualization

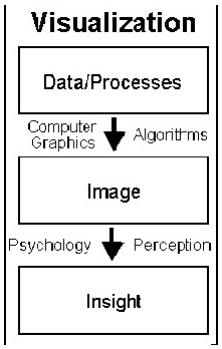
Most important aspect in visualization, is that the interface, or access, between the visualization tool and the stored captures is flexible enough to accept any format of data, to include data from multiple sources. It must also do this without seriously inhibiting the timeliness of the rendered output. If the tool is too cumbersome or resource intensive then you may limit the amount of manipulation that is possible with the rendered result. The capability of manipulating the data is the key to making a visualization technique an integral and useful part of monitoring and reporting.[24]

The idea behind data visualization in monitoring tools analysis is that the data may be presented to the user in a format that is optimized for ease of comprehension, and to make data and patterns more easily recognizable. A prime benefit of being able to visualize these captures is that the new perspective often lends itself to revealing hidden patterns that may not be readily apparent from the context of a flat file or queried result. Also, the efficiency with which we can perform analysis on large amounts of data can be increased,thus maximizing those resources required when performing that analysis. Therefore, designers constructing capture and access applications are faced with more than just issues related to different pieces of data. Beyond data, there are still the users, the devices, time and locations involved in the experience to take into consideration in the design.[25]

Furthermore, a true science of visualization must incorporate both a formal theory of computer graphics and a theory of human perception.

One of the problems that plague data exploration and analysis involves large amounts of data that can be difficult to visualize in ways that dont overwhelm the viewer or hide whats important behind a wall of clutter. This is a problem that has been receiving a great deal of recent attention by the research community. Methods are being explored and sophisticated algorithms are being developed to tame the quantity of data either by reducing the amount in ways that avoid loss of meaning, or by reducing visual

Figure 3.1



clutter in the visualization itself through novel approaches to the positioning of data objects, better uses of color, or other visual attributes such as transparency. This work is ideal for being included in commercial visualization software that is otherwise already effective.[26]

There are mainly three important issues in data visualization.[24]

Resource

Increasing the amount if monitoring usually leads to increase in resources such as storage and processing power as well. Most monitoring system generate huge amount of data and persisting these data's for historical analysis or pattern recognition demands lots of storage space to persist these data which can become a huge problem with respect to management of the data and scalability issues as well. The more data we have for visualization more resource are needed for the data processing.

Integration and Interoperability

Most tool in the network system co-exist together performing similar operation and gathering similar type of data. Besides the issue of accessing the data is that of defining and representing the data in a meaningful structure. Many data source exist such as flat format , database but most of them can be ported or customized to the requirements in constraint of cost and feasibility

Human factor

How people perceive and interact with a visualization tool can strongly influence their understanding of the data as well as the system's usefulness. Human factors therefore contribute significantly to the visualization process and should play an important role in the design and evaluation of visualization tools. [27] Much of the current methodology for designing visualization tools and interfaces is ad hoc and informal. Only a few visualization designs utilize perceptual and cognitive theories.

For human beings, our potential is directly constrained by our attention, memory, and processing capabilities.[28] We human suffer from difficulty dealing with and processing information that exists visually in more than three dimensions. Thus, many

tools are governed by what is assumed to be the ability of the user for useful perseverance.

Chapter 4

Model and Methodology

4.1 Defining a common model for data representation

Integrating data among different tools and making it accessible for analysis and integration for another system has major benefits. The goal of integrating or extracting information from different source is to have a uniformed, well defined global view of the information which can be readily accessible by the user and can be uniformly queried with a well defined interface which abstracts the underlying heterogeneous data source.

Initial step of accessing data and integrating with another source requires defining a common model through which every sources data can transformed with semantically rich representation of the source data.

The use of data transformation and integration for addressing the problem of interoperability between heterogeneous information systems has been studied before [[29],[30],[31]]. But in practice most of the tools has not been updated or build to exchange data easily for reuse.

Data Sources and Services are usually independent of each other. As a result they have their own structures and/or schema's which may be very different. A common data model provides a model that spans the entirety of a target domain applications and data sources and effectively overlays the multitude of databases which data is drawn from by individual applications. At the core of a common data model is the need for all the data relationships, terminology and meanings that exist within an organization to be clearly defined, thereby enabling a carrier to map its existing applications and data definitions to the common model.

To represent such a common model we have to know the underlying system data between different sources and try to come up with a rich representation which encloses all the necessary data and context.

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data.[32].

In many cases of mutually inconsistent data sources, the problem is generally dealt with by means of suitable transformation and cleaning procedures applied to data retrieved from the sources.

Mapping between different information sources is a key step in integration approach.

Currently there are mainly two major ways to create and manage mappings

4.1.1 Global-as-view GAV

In this approach, the integration is done by creating a middle integrated schema or a global view which encloses all the source data views into single schema where data are integrated and queried upon. This idea is effective whenever the data integration system is based on a set of sources that is stable. The GAV approach favors the system in carrying out query processing , because it tells the system how to use the sources to retrieve data. In this case users issue queries against the mediated schema. However, GAV suffers from two some shortcomings. It is very difficult to remap or rewrite the global schema once any new source with new information is added. So updating the global schema becomes tedious. And is very difficult to come up with a complete data that sources may include. 4.1 illustrates an example of GAV architecture.

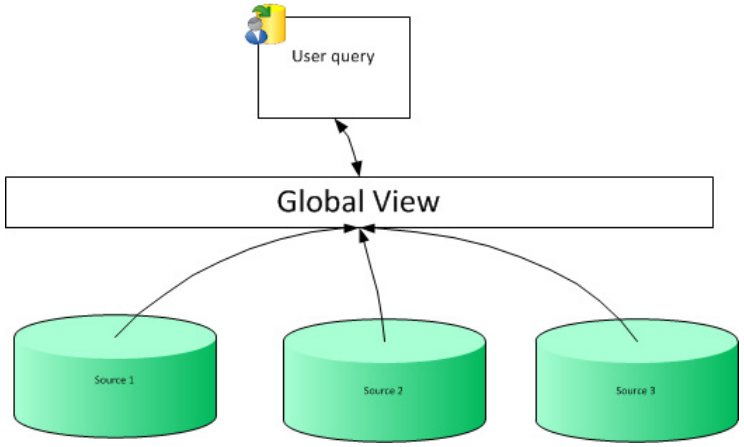


Figure 4.1: Global As View example

4.1.2 Local-as-view LAV

In this approach the mediator schema is just a domain model, as an conceptual database comprising of information on each source. A source profile describes which portion of the worlds information the source has. A new type of query processor matches a user request against the currently available sources, and devises a query that computes the desired result. The approach is often called profiling or Local As View (LAV); when we use it as a component of a larger mediation approach, we will sometimes refer to it as downward view (from M-schema to sources). This idea is effective whenever the data integration system is based on a global schema that is stable and well-established in the organization. LAV approach favors the extensibility of the system: adding a new source simply means enriching the mapping with a new assertion, without other changes. 4.2 illustrates an example of LAV architecture.

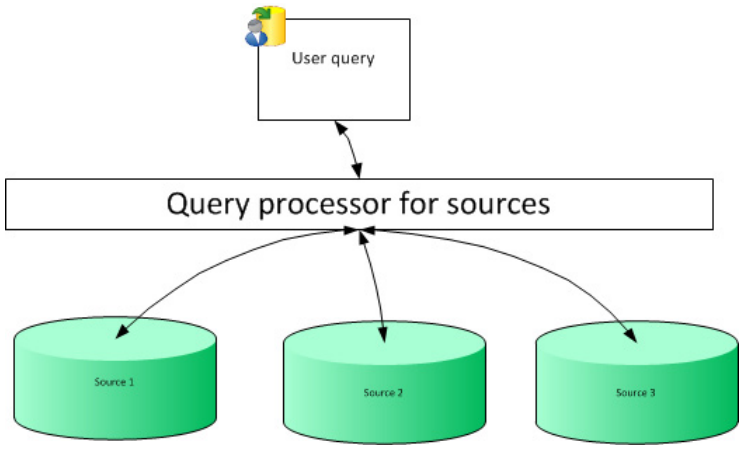


Figure 4.2: Local As View example

4.2 XML as common data model based on data to be integrated

For our purpose we have chosen two different application CFEngine [17] and Munin [33] as primary client source form which data are collected and two source application where the data's are being integrated,Graphite [3] and Groundworks[2].

Each of these tools gather basically two categories of data

- Host specific data such as name,ip etc
- Performance data such as disk free, number of users , no of processes etc

So studying these data model that they each capture and store for their processing and the data model that is needed for the integration to the required source i.e in our case Groundwork and Graphite. A simple common model can be generated which expresses the data collected by the tools that can be easily exchanged and manipulated. Basically the tool collect discrete time-series related performance data for monitoring purposes which can be easily represented in an XML structure having schema as in 4.1

Listing 4.1: XML schema of the common model

```
<xs:schema
attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="host">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:int" name="time"/>
        <xs:element type="xs:string" name="status"/>
        <xs:element type="xs:string" name="ip"/>
        <xs:element type="xs:string" name="name"/>
        <xs:element type="xs:string" name="source"/>
        <xs:element name="service" maxOccurs="unbounded" minOccurs="0">
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:byte" name="Max"/>
              <xs:element type="xs:string" name="Unit"/>
              <xs:element type="xs:byte" name="Warn"/>
              <xs:element type="xs:int" name="MeasuredTime"/>
              <xs:element type="xs:string" name="State"/>
              <xs:element type="xs:string" name="Message"/>
              <xs:element type="xs:byte" name="Min"/>
              <xs:element type="xs:string" name="Label"/>
              <xs:element type="xs:byte" name="Critical"/>
              <xs:element type="xs:byte" name="MeasuredValue"/>
              <xs:element type="xs:string" name="Type"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.2.1 Why XML as data source?

XML was introduced in the mid-1990s as a mark-up language which defines a set of rules for representation of document in a format that is both easily readable by humans and easily processed by machines. It gained immediate foothold as a data interchange format. Over time, gaining from a wide variety of research, XML has become a valuable data format within and across enterprises for representing data in persistent and transient applications.

XML provides a quite natural way of structuring data, based on hierarchical, graph-based representations. Such models have the advantage of being simple, standard, and well accepted, but also powerful enough to represent structured, unstructured, and semistructured information. Thus, XML works well as a common data model in variety of different scenarios be it for different applications or standard web. In addition, several research efforts have investigated the use of graph-based data models for managing semistructured data and for integrating heterogeneous data sources.

Standardizing the schema and the tags of XML greatly increases the portability of the data and also help in simplifying the data integration process. Furthermore, XML enables one to more naturally model differences between representations of data in different sources.

Another major advantage of using XML as common format is the availability of the tools and support for processing it. Almost every programming platform or application has some level of XML processing capability which makes it very easy for processing XML based data.

4.2.2 Definition of common model elements

As listed on 4.1 the schema of common model consist of the data structure that comprises both essential elements from the source data and the elements needed to integrate the data to another system.

Basically the XML schema tries to model two types of information specifically, one is host information and another is the service information that it is monitoring. The illustration in 4.3 gives a overview of the data thats being modeled.

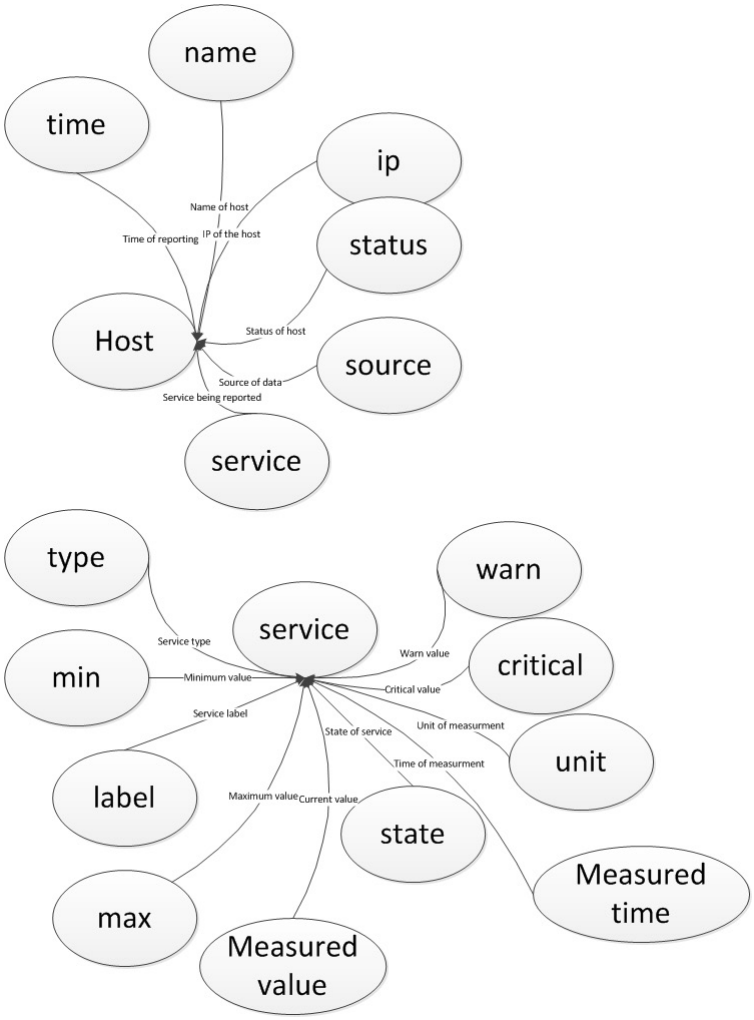


Figure 4.3: Diagram of common schema

The Host class models the attribute of the host that the data is currently being analyzed which consist of

name A hostname is a label that is assigned to a device connected to a computer network and that is used to identify the device.

ip This denotes the ip address of the host

status This indicated the current status of the host while reporting. Can be any string denoting the current status as OK,DOWN,UP etc

time Timestamp of when the data is reported of the host

source The application source from where the data is generated eg cfengine,munin etc

The service class models the attribute of the services or performance parameter for which the host is currently reporting.For example it can be cpu, loadavg, no of users, no of processes, memory etc. It consist of current attributes

min Minimum threshold value of the performance parameter

max Maximum threshold value of the performance parameter

status This indicated the current status of the service while reporting. Can be any string denoting the current status as OK,CRITICAL,WARNING,DOWN,UP etc

measuredTime Timestamp of when the data is reported of the service

warn Value at which a warning can be generated

critical Value at which service is indicated to reach the critical level

measuredValue Current measured value of the service

unit unit of currently measured value

type unique identifier for the service

label General description of the service

Message Extra information about the service reporting

4.3 Architecture for integration

Once we have a common model defined, next step is to extract or transform the data from the different source into the common model and then again analyzing the common model to export the data to another format for the target system integration. figure 4.4 gives a general overview of the architecture of the method for the data integration.

The architecture contains two layers of processes to transform or integrate one source of data to another. In the following, brief description about the layers and their services are described

1. Data wrapper layer As the source data has different structure that they are stored , they need to be transformed to our common XML model that has been described. In order to achieve this goal , wrapper for different source has to be in place which can read the data provided by the source and then transform it to our common model. So in general every data source would have a wrapper around it which converts the data. The wrapper can be a general script or program which manipulates the source data and assimilates any information that needs to be integrated to the target systems. Example of this would be a program which collects data from CFEngine or Munin which has to be integrated to the other system which maybe groundwork or graphite or any other system and then formats the data collected into our common XML schema.

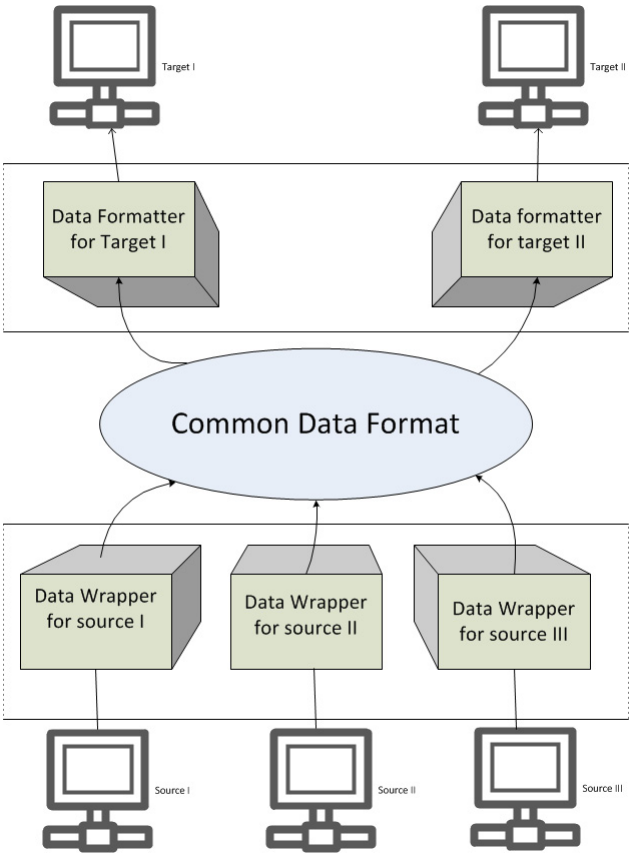


Figure 4.4: General Architecture View

2. Data formatter layer Once the data that has been formatted to our common schema by the wrapper layer, It has to be integrated to other systems which expects the data in different format. So the main function of the formatter layer is to format the common data model into the data format that the target system expects. There would be different formatter for different target system. For example if we are integrating data into the Groundwork then the formatter would format the data in the format which groundwork expects. If we need to integrate to any other system then similar formatter is created to carter the data into the expected formated.

The benefit of having such a architecture is that each of the component remains loosely coupled and can be developed separately and as needed. And they behave as plug and play system. For example if any new source is added from which data has to be integrated , a new data wrapper can be created easily to export the data into common model and the same old formatter would suffice to integrate into the target system. Also having a common model that is flexible enough to contain required data makes it easier to transform it into many different target source without having to modify the targeted applications.

Chapter 5

Implementation

For the implementation of our proposed architecture for data integration, we have CFEngine and Munin as our primary data source from which the data collected are then integrated into the target systems which are Groundwork and Graphite.

5.1 Data wrapper layer for CFEngine

The primary purpose of the data wrapper layer is to collect the data from the source database or application and then formatting it to the common model. For this to happen we must know the applications structure or API through which the data we need can be extracted.

Cfengine is a policy-based configuration management system originally written by Mark Burgess. Its licensed under the GNU General Public License (GPL) . It uses a declarative language to define the desired end state of a system, as opposed to some other configuration management tools that define what should be done to a system. In CFEngine the state of the machines are managed through promises written in policy files that get distributed to each agents running on the system.

It was first released in 1993, and it has evolved over time with evolving configuration and monitoring scenarios , such as virtualization and cloud computing. CFEngine is developed and designed to make it possible to automate very large numbers of systems in a scalable and manageable way. It uses very less resource compared to other comparative tools, and it can run on everything from embedded devices and smart phones to supercomputers.

New version of Cfengine i.e CFEngine 3 was released in 2008. CFEngine 3 is different from many other automation mechanisms in that you do not need to tell it what to do. Instead, you specify the state in which you wish the system to be, and CFEngine will automatically and iteratively decide the actions to take to reach the desired state, or as close to it as possible. Underlying this ability is a powerful theoretical model known as Promise Theory[34], which was initially developed for CFEngine 3, but which has also found other applications in Computer Science and in other fields such as Economics and Organization.

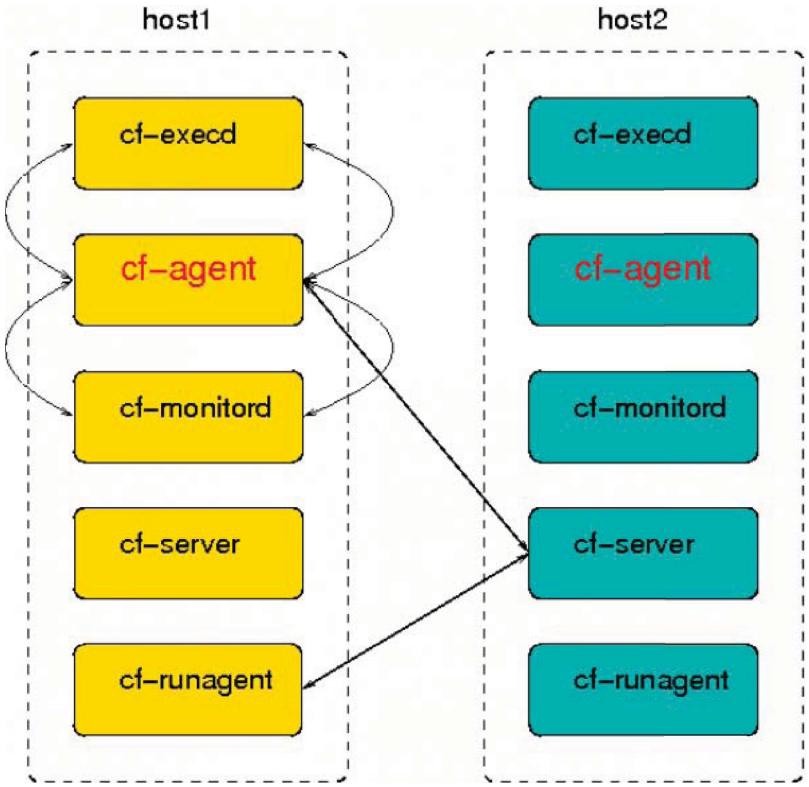
Although the prime purpose of CFEngine is not monitoring, it contains one of the most flexible and lightweight monitoring engines around. You can extract data about system configuration, usage, resources and log data and turn this into readable reports.

CFEngine's has ability to monitor and extract useful monitoring performance matrices which servers it as a very capable monitoring and reporting tool.

5.1.1 Architecture

CFEngine is an extensive framework. CFEngine runs as a client/server application. The server provides new configurations, or policy files to the client, while the client works to ensure the policy that has been specified. The policy is written the descriptive language of CFEngine. This language describes the promises or state that host should maintain. The agent makes the client conform to this policy in a convergent manner by iterating over the policy repeatedly. CFEngine's software agents run on each individual computer but can communicate if they need to, as depicted the figure 5.1.

Figure 5.1: Components of CFEngine *source:www.cfengine.com*

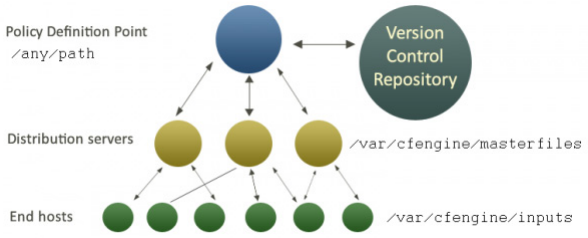


The figure 5.2 shows how decisions flow through the parts of a system.

CFEngine is a pull based system. Every client on CFEngine system pulls the policy files from the policy hub. Usually there is only one policy hub but can be replicated to have better load balance if required. It is agent based software. It resides on and runs processes on each individual computer under its management. So every computer that needs to be managed must have CFEngine agent running on it and must have one policy server through which it can connect and pull the policies to execute on the system.

CFEngine uses a simple, private protocol that is based on (but not identical to) that used by OpenSSH (the free version of the Secure Shel). It is based on mutual,

Figure 5.2: decision flow of CFEngine *source:www.cfengine.com*



bi-directional challenge-response using an autonomous Public Key Infrastructure.In CFEngine authentication by Public Key is mandatory and encryption of data transfer is optional.

5.1.2 Monitoring and Reporting

In CFEngine, cf-monitord is a process which runs periodically every couple of minutes and samples data for a number of performance matrices. The data are then stored in an embedded database on the local host, using a algorithms that prevents the data size from growing endlessly just like RRDtool.CFEngine learns about the systems pattern and behavior over the period of time and reports about the measurement based on the past behavior of the data. If something is out of the normal state that has been observed over period of time then the alerts and reporting are done based on the classes that CFEngine uses. CFEngine[17] is not a comprehensive monitoring tools so there are certain goals behind the monitoring solutions of CFEngine. Some of those goals are

- To not waste users' time with insignificant changes, but provide meaningful updates at a rate that is defensible based on the rate of change of the system.
- provide meaningful information that is placed in the context of what is normal.
- reveal trends and patterns at a glance.
- To scale to tens of thousands of hosts without placing a significant burden on the hosts being monitored.
- To be as hands-free in configuration as possible, but allow customization.
- To provide a feedback mechanism for system policy so that systems can respond directly to conditions that are detected.

The list of measured attributes that is currently being monitored by CFEngine are fixed to the following variable:

users The number of different users that appear in the process table of the system.

rootprocs The number of current processes started by root/Administrator.

userprocs The number of current processes started by non-privileged users.

diskfree The amount of disk free on root file system.

loadavg The load average of the system (actually multiplied by 100).

Socket counts of network services distinguish between incoming and outgoing sockets (to a service or from a client).

netbiosns Registers traffic to/from port 137.

netbiosdgm Registers traffic to/from port 138.

netbiosssn Registers traffic to/from port 139.

irc Registers traffic to/from port 194.

CFEngine Registers traffic to/from port 5308.

nfsd Registers traffic to/from port 2049.

smtp Registers traffic to/from port 25.

www Registers traffic to/from port 80.

ftp Registers traffic to/from port 21.

ssh Registers traffic to/from port 22.

wwws Registers traffic to/from port 443.

If tcpdump program installed in a standard location, then the monitor can be configured to collect data about the network flows to your host.

icmp Traffic belonging to the ICMP protocol (ping etc).

dns Traffic to port 53, the Domain Name Service (usually a special case of UDP).

udp Miscellaneous UDP traffic that is not related to DNS.

tcpsyn Registers TCP packets with SYN flag set.

tcpack Registers TCP packets with ACK flag set.

tcpfin Registers TCP packers with FIN flag set.

misc Registers all other packets, not covered above.

The above description and variables are mentioned on [35].Reporting capabilities in CFEngine depend on the software version include:

Community Edition Basic output to file or logs may be customized on a per-promise basis. Users can design their own log and report formats, but data processing and extraction from CFEngine's embedded databases must be scripted by the user.

Nova In addition to community features, Nova provides automated extraction of data from CFEngine's self-learning agents, and the generation of a standard set of reports in text, HTML or XML formats. Nova summarizes distributed data and provides simple compression and aggregation of these summaries. Finally summaries are tied into a knowledge map or semantic index for browsing by IT operations. Command line tools in cf-report are also available for Nova users to browse network-wide data.

CFEngine's default behavior is to report to the console (known as standard output). It's default behavior is to report nothing except errors that are judged to be of a critical nature. But CFEngine allows you to customize your report generation using "reports" promises. Where we can customize the reports to any format like log, html, xml etc. It interfaces with the system logging tools as well like Syslog for Unix-like systems, while the event logger on Windows.

5.1.3 Data storage model

CFEngine approaches monitoring and reporting from the viewpoint of scalability so there is no default centralization of reporting information, as this is untenable for more than a few hundred hosts. So every host retains it own data.

In CFEngine the storage of the collected data values or monitored probes are stored in a time series data in which an iterative algorithm is applied to analysis on the stored periodic time to provide a smooth scroll-off in the significance of the data with time. The Algorithm applied for the storage not only reduces the storage space for the data by compressing it , but also decreases the computation cycle compared to traditional time-series approach. The work is fully summarized in the paper "Two dimensional time series for anomaly detection and regulation in adaptive systems" [36].

The data are gathered for a week at the interval of five minutes and are stored in embedded database with simple key value pair of format "day:hour:Minute_interval".Currently the supported database systems are Tokyo cabinet [37] and qdbm [38].

5.1.4 API

API stands for Application program interface. It is used for interfacing one program with another program so that the interaction between multiple program would be easier. An API can be of many form ranging from complex library for programming language or simple protocol for accessing certain features of the program. Some of the example of API includes WEB API which includes set of HTTP protocol or REST style API [39],Java APIs etc.

One of the most important factor for any program data to be manipulated or utilized is to have access to the data it stores. This can be achieved through many different ways. Also having a good API for accessing the underlying data means having a better chance of integrating with other software platforms.

In CFEngine community edition the standard way of getting the data out for integrating with other tool is with cf-report utility. It can generate the monitored data in the format that is described in the policy i.e (xml,html,csv or plain text) through which one can collect the data's and manipulate it. It takes data stored in CFEngine's embedded databases and converts them to human readable form.

Another way of getting the performance data from cfengine is to use the internal policy variables. The variables discovered by cf-monitord are placed in 'Variable context "mon". Monitoring variables are expected to be ephemeral properties, rapidly changing.

5.1.5 Implementation

First approach

First approach of getting the data out of the cfengine was to use the cf-report utility to generate the csv file with all the monitored data and information and then using a Perl script to parse the generated file and model the data as we need and pass it to the formatter for the integration with the target system. cf-report utility takes data stored in CFEngine's embedded databases and converts them to human readable form. cf-report keeps the promises made in common bundles, and is affected by common and reporter control bodies. Part of the policy where the generation of report is specified is described in the listing 5.1

Listing 5.1: CFEngine policy for report generation

```
body reporter control
{
 any::

  reports => {
              "all"
              };

  build_directory => "$(sys.workdir)/reports";
  report_output   => "csv";
  style_sheet => "/cf_enterprise.css";
```

So whenever we run the cf-report utility in command line it will produce the csv files of monitored data in the reports directory which has format as listing 5.2

Listing 5.2: CFEngine summary report generation file

```
 0,users      ,   4.200000,   5.994168,   0.100000
 1,rootprocs  ,  47.600000,   1.000000,   0.100000
 2,otherprocs ,   3.500000,   1.000000,   0.100000
 3,diskfree   ,  62.300000,   1.000000,   0.100000
......
```

Where the 2nd column denotes the service thats beings measured and 3rd column has the value of the service determined at that time. So a simple perl script to read the generated file and extract the service information and its measured value is developed which after reading the values converts the data into XML format that conforms to our common data model. The state of the services can also be deduced based on the classes that are set during the monitoring activity. They are based on the anomaly detection system that CFEngine uses[36]. CFEngine classifies anomalies by whether the currently measured state of the system is higher or lower than the average for the

current time of week. The amount of deviation is based on an estimate of the 'standard deviation'. When CFengine has acquired enough monitoring data, it classifies the current state of the monitored metric into 4 levels:

normal means that the current level is less than one standard deviation above normal.

dev1 means that the current level is at least one standard deviation about the average.

dev2 means that the current level is at least two standard deviations about the average.

anomaly means that the current level is more than 3 standard deviations above average.

Each of these characterizations assumes that there are good data available. The cf-monitord evaluates its data and decides whether or not the data are too noisy to be really useful. If the data are too noisy but the level appears to be more than two standard deviations above average, then the category microanomaly is used.

Some example classes that are set in the context are userprocs_high_dev2, userprocs_low_dev1, www_in_high_anomaly. So based on these classes we can determine the state of the services and this class information can be read from the classes.csv file that cf-reports produces when it runs.

Sample of generated class file is given at listing 5.3

Listing 5.3: CFEngine class file

```
1.0000, 0.0000,10_0_0_6,Tue May  1 17:4
1.0000, 0.0000,users_high,Tue May  1 17:4
0.0156, 0.1754,cfengine_out_high,Tue May  1 17:3
0.0156, 0.1754,otherprocs_high,Tue May  1 17:3
1.0000, 0.0000,entropy_misc_out_low,Tue May  1 17:4
1.0000, 0.0000,entropy_misc_in_low,Tue May  1 17:4
1.0000, 0.0000,diskfree_high_normal,Tue May  1 17:4
0.9988, 0.0449,cpu0_low,Tue May  1 17:4
0.9988, 0.0449,cpu_low,Tue May  1 17:4
```

The perl script generates the desired XML format and then send it to the Formatter through socket which then is further processed. The script can be run as a cron job with specified interval suitable for reporting the data. The data format generated by the script is shown below(truncated for display). Full source code of the implementation is shown at listing A.1

Listing 5.4: cfengine-data-collector.pl

```
<Host>
  <ip>10.0.0.6</ip>
  <name>cfengine</name>
  <source>CFEngine</source>
  <status>OK</status>
  <time>1336086854</time>
  <service>
    <Critical></Critical>
    <Label>users info</Label>
    <Max></Max>
    <MeasuredTime>1336086936</MeasuredTime>
```

```
    <MeasuredValue>4.200000</MeasuredValue>
    <Message>
    users checked, max-val:: 5.994168 and min-val::4.200000|users=4.200000;0;0;0
    </Message>
    <Min></Min>
    <State>OK</State>
    <Type>users</Type>
    <Unit>float</Unit>
    <Warn></Warn>
  </service>
  <service>
    <Critical></Critical>
    <Label>rootprocs info</Label>
    <Max></Max>
    <MeasuredTime>1336086936</MeasuredTime>
    <MeasuredValue>47.600000</MeasuredValue>
    <Message>
    rootprocs checked, max-val:: 1.000000 and min-val::47.600000|rootprocs=47.600000;0;0;0
    </Message>
    <Min></Min>
    <State>OK</State>
    <Type>rootprocs</Type>
    <Unit>float</Unit>
    <Warn></Warn>
  </service>
  <service>
    <Critical></Critical>
    <Label>otherprocs info</Label>
    <Max></Max>
    <MeasuredTime>1336086936</MeasuredTime>
    <MeasuredValue>3.500000</MeasuredValue>
    <Message>
    otherprocs checked, max-val:: 1.000000 and min-val::3.500000|otherprocs=3.500000;0;0;0
    </Message>
    <Min></Min>
    <State>OK</State>
    <Type>otherprocs</Type>
    <Unit>float</Unit>
    <Warn></Warn>
  </service>
  <service>
    <Critical></Critical>
    <Label>irc_in info</Label>
    <Max></Max>
    <MeasuredTime>1336086854</MeasuredTime>
    <MeasuredValue>0.000000</MeasuredValue>
    <Message>
    irc_in checked, max-val:: 1.000000 and min-val::0.000000|irc_in=0.000000;0;0;0
    </Message>
    <Min></Min>
    <State>OK</State>
    <Type>irc_in</Type>
    <Unit>float</Unit>
    <Warn></Warn>
  </service>
</Host>
```

Second approach

Another method of getting the monitoring value out of the CFEngine is to use its internal policy and access the variable defined during the agent run. CFEngine stores the monitoring data in the context of mon and can be accessed any where in the promise which can be then utilized for different purpose. This method is more reliable and conforms more with CFEngine way than the first approach that is defined. For this implementation a CEEngine policy is written to report any needed performance metric through command promise to execute our custom perl script with the performance value and other information as a parameter which then is analyzed and then modeled into XML format and then sent to the formatter.

Below is the policy written in CFEngine where it reports the performance data to external script.

Listing 5.5: cfengine policy to report performance data

```
bundle agent monitor
{
  vars:
  "cfengine_observables"  slist  => {
                                    "diskfree",
                                    "users",
                                    "rootprocs",
                                    "otherprocs",
                                    "loadavg",
                                    "cfengine_in",
                                    "cfengine_out",
                                    "ssh_in",
                                    "ssh_out",
                                    "cpu",
                                    "syslog"};


 commands:


  "/usr/bin/perl /root/data-collector.pl -s OK
  -m $(cfengine_observables) -c $(mon.value_$(cfengine_observables))
  -n $(sys.host) -i $(sys.ipv4)"
  ifvarclass => not(canonify("$(cfengine_observables)_high_dev2)"));

  "/usr/bin/perl /root/data-collector.pl -s Warning
  -m $(cfengine_observables)
  -c $(mon.value_$(cfengine_observables))
  -n $(sys.host) -i $(sys.ipv4)"
  ifvarclass =>  canonify("$(cfengine_observables)_high_dev2");


  "/usr/bin/perl /root/data-collector.pl -s Critical
  -m $(cfengine_observables) -c $(mon.value_$(cfengine_observables))
  -n $(sys.host) -i $(sys.ipv4)"
  ifvarclass => canonify("$(cfengine_observables)_high_anomaly");
```

}

In above policy "cfengine_observables" is the list of observables or services to report data on. And the command promise executes a perl script which takes the parameter with service status, service name, service value, host and ip. Service status are differentiated with the class context. eg if users_high_dev2 class is set for the users service then service status is set as warning where as if users_high_anomaly is set then critical status is sent as service status. The external perl script when executed with the defined parameter generate the same XML structure as in listing 5.4 and send the xml file to formatter for further processing. Full source code of the implementation is shown at listing A.2.

5.2 Data wrapper layer for Munin

Munin is a networked resource monitoring tool that can help analyze resource trends and detect performance problems.It collects different matrices form different configured nodes or hosts and then persists it on central server for analyzing and viewing graphs.Its emphasis is on plug and play capabilities. Using Munin you can efficiently and easily monitor the performance of your computers, networks, applications and whatever comes to mind. Munin uses the RRDTool (written by Tobi Oetiker) and the framework is written in Perl, while plugins may be written in any language.

5.2.1 Architecture

Munin has a master/node architecture in which the master connects to all the nodes at regular intervals and asks them for data. It then stores the data in RRD files, and (if needed) updates the graphs. One of the main goals has been ease of creating new plugins (graphs). Munin is run by a cron job every five minutes. So, every five minutes, it connects to all the servers via a very simple protocol and plain TCP that it has to monitor, fetches all the data, writes the data in RRD files, and recreates all the HTML files and hundreds of PNG files.

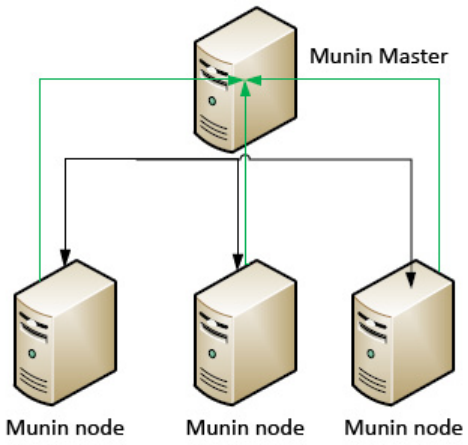


Figure 5.3: Munin Architecture

5.2.2 Monitoring and Reporting

Munin collects all the monitoring data with its plugin system. Munin plugins run on all the nodes that Munin monitors and report back all kinds of different statistics that Munin can graph later. Plugin are generic and can be added to measure whatever one wishes. Each munin-node installation brings an initial set of plugins. Depending on the purpose and equipment of the node, we can add further plugins to monitor special services or hardware. It updates the data at an interval of every five minutes by default.

Munin reports all the collected data by generating graphs and showing it on a graphical web interface written in PHP.

5.2.3 Data storage model

Munin uses RRDTool [40] for both data logging and graphing system.Every things that munin-master collects through the plugin is stored in rrdtool and then graphs are generated in regular interval which are accessible from the web interface.

5.2.4 API

Munin does not have any official API to get the required data out. But as all data are stored in RRDtool,we can manipulate the database and get the data out for manipulating. Another way to get the data out is to use the protocol used by the munin daemon which it uses to talk to its nodes. Protocol for data exchange between daemon and client is very simple and is described below

help Show available commands

list (node) Asks client to list all query-items available for query for this host. If no host is given, default to host that runs the munin. Example load, cpu, memory, disk etc

node nodes List hosts

config (query-item) Asks the client for configuration items

Listing 5.6: Example of munin config command

```
> config load
< graph_args --title "Load average"
< load.label Load
< .
> config memory
< graph_args --title "Memory usage" --base 1024
< used.label Used
< used.draw AREA
< shared.label Shared
< shared.draw STACK
< buffers.label Buffers
< buffers.draw STACK
< cache.label Cache
< cache.draw STACK
< free.label Free
```

```
< free.draw STACK
< swap.label Swap
< swap.draw STACK
```

fetch (query-item) Fetches actual values.

Listing 5.7: Example of munin fetch command

```
> fetch load
< load.value 0.42
< .
> fetch memory
< used.value 98422784
< shared.value 1058086912
< buffers.value 2912256
< cache.value 8593408
< free.value 235753472
< swap.value 85053440
```

version Print version string

5.2.5 Implementation

Mapping of Munin data to common model

A data wrapper for Munin was implemented using a perl script which would utilize the munin's protocol to talk with munin nodes using sockets and then fetch all the parameters that are measured and their value and then transform those collected parameters into our common data model i.e the XML format and send it to the formatter for further processing. The script can be scheduled with a cron job or invoked by any other tools as required. Usually by default the munin nodes listen to the commands at port 4949. So if we send the above listed protocol command to munin nodes then we are able to collect the data and other required parameters for our processing. Full source code of the implementation is shown at listing A.3. Sample truncated output of the data this wrapper generates is listed at 5.8.

Listing 5.8: data output of munin wrapper

```
<Host>
    <ip>10.0.0.5</ip>
    <name>munin</name>
    <service>
      <Critical></Critical>
      <Label></Label>
      <Max></Max>
      <MeasuredTime>1336230485</MeasuredTime>
      <MeasuredValue>313065</MeasuredValue>
      <Message>OK system user current value 313065</Message>
      <Min></Min>
      <State>OK</State>
      <Type>system.user</Type>
```

```
    <Unit>float </Unit>
    <Warn></Warn>
  </service>
  <service>
    <Critical></Critical>
    <Label></Label>
    <Max></Max>
    <MeasuredTime>1336230485</MeasuredTime>
    <MeasuredValue>812179</MeasuredValue>
    <Message>OK system nice current value 812179</Message>
    <Min></Min>
    <State>OK</State>
    <Type>system.nice</Type>
    <Unit>float </Unit>
    <Warn></Warn>
  </service>
  <service>
    <Critical></Critical>
    <Label></Label>
    <Max></Max>
    <MeasuredTime>1336230485</MeasuredTime>
    <MeasuredValue>445589</MeasuredValue>
    <Message>OK system system current value 445589</Message>
    <Min></Min>
    <State>OK</State>
    <Type>system.system</Type>
    <Unit>float </Unit>
    <Warn></Warn>
  </service>
  .....
```

5.3 Data formatter for Groundworks

Next phase after the modeling of the source data into common data schema , it is to transform it to the format that the target system expects it to be. Here we take two software as our target system where the collected data will be integrated , they are i) GroundWork and ii) Graphite.

The implemented data formatter runs as a daemon listening to any incoming message and when upon receiving the message from the wrappers, tries to covert it to the targeted system format for integration.

Groundworks (GWOS) is an unified monitoring and network management tool which is built upon existing tools such as Nagios, NMap , RRDTools etc to give a integrated one system administration tool to monitor anything. Its basically a system to integrate multiple tools that can gather data and assimilate those data into one common domain and present it in one place. GroundWork gives you availability and performance information, with visualization tools and graphing. The included portal makes it easy to integrate existing IT operations tools, like inventory, ticketing, asset and configuration management systems. The GWOS integrates Nagios, SNMP protocols, RRDtool, JBoss Portal, ICEfaces, MySQL, BIRT, Ganglia and Cacti which are mature open source tools and widely deployed in various installation.

5.3.1 Architecture

GWOS is mostly built upon existing open source technologies. These technologies includes JBOSS, Apache, Nagios, Cacti, and MySQL etc.

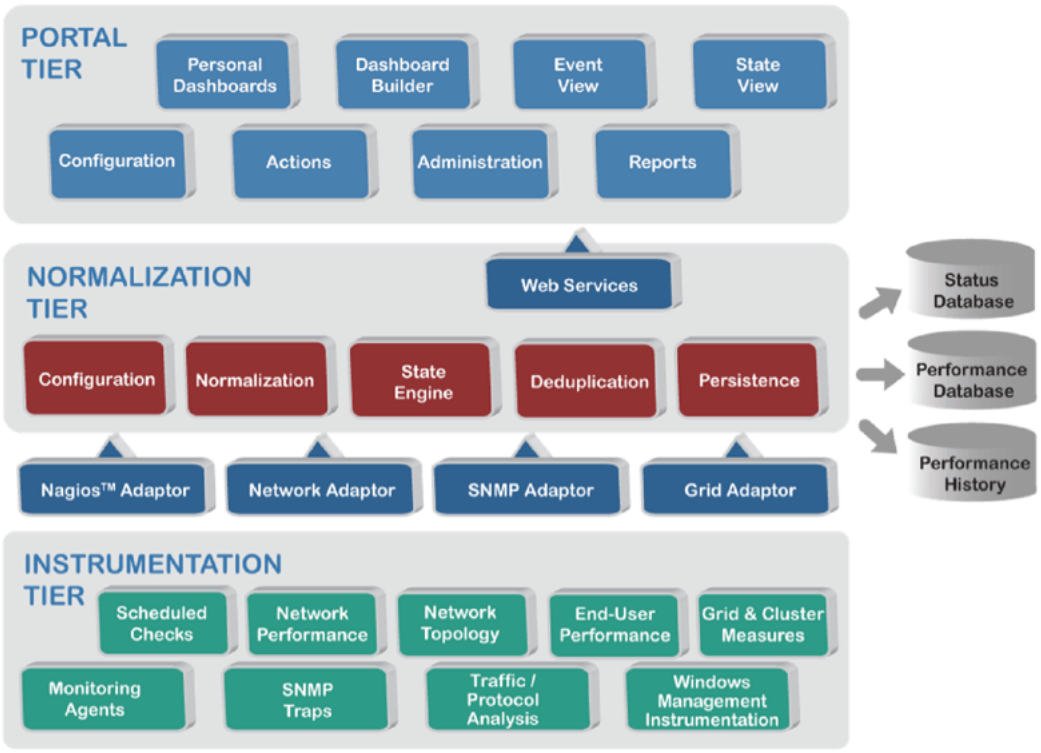


Figure 5.4: GWOS Architecture *source:http://www.gwos.com/technology/*

Tier 1: Instrumentation This is the layer that utilizes the most varied components. Data gathering is performed by Nagios, by Cacti, and by other optional components that use a standard, well documented data feeder for their output. Data is captured as state changes, events, and performance measures.

Tier 2: Normalization Essentially, this layer stores the data in a normalized form, and presents it on demand through web services or database queries. Think of it as the data foundation of the next tier.

Tier 3: Portal This is the UI part of the system where users can see most of the reports, status , graph and different system configuration of the system.

5.3.2 Monitoring and Reporting

GroundWork(GWOS) is equipped to monitor dozens of common infrastructure elements, primarily Nagios. Anything that can be monitored by Nagios is available in GWOS. But it is not limited to Nagios, any other tool can be integrated for monitoring.

The configuration of this part of GroundWork is done by collecting commonly checked metrics into Profiles, which can then be applied to hosts. A profile can define the monitoring of anything, leveraging the open nature of the system. The target can be a server, a network device, an application, database or web server, or even an abstract collection of other objects.

Groundwork has rich web portal through which various information and reports can be accessed. Logs of events, performance graphs and many other information are available through the web portal.

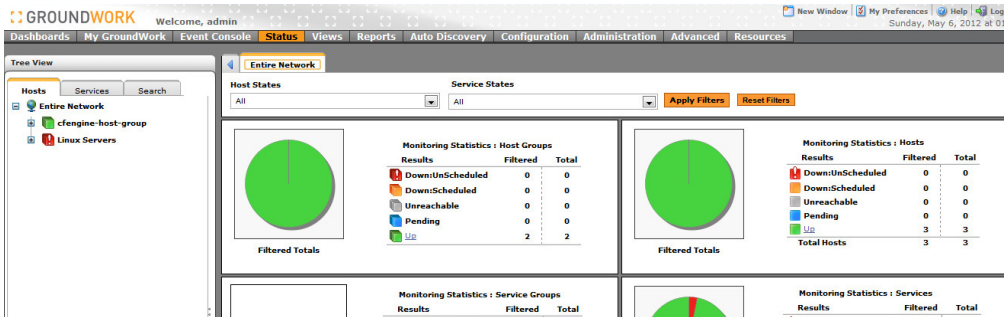


Figure 5.5: Groundwork web interface

5.3.3 Data storage model

All the data that are collected by groundwork are stored in the rational database system. Current version of groundworks support PostgreSQL [41] database and MySQL [42] database as their storage system.

In GroundWork storage of performance data is stored and manipulated by open source RRD toolkit. The per script that comes with Groundwork periodically scans Nagios logs file for performance data and populates the RRD database as configured on the system. Any performance data that are submitted to Nagios can be graphed as well as non Nagios specific data can be configured for storing and graphing as well.

To created RRD database for reporting, it provides a separate ”Performance Configuration” section where many parameters for creating,updating graphs can be adjusted. This part of application allows the administrator to configure the services that they want to have the graph from RRD database.

5.3.4 API

The model of GroundWork Foundation is closely related to the Nagios objects. This includes:

Host Groups This includes Hosts as members.

Hosts This typically represents physical devices, and includes Services.

Service This typically represents a Nagios plugin executing on a specific host. A Host-Service combination is unique in the monitoring system.

The following type of information can be retrieved through the standard API supplied by the system.

Host Status This represents the current status and attributes of Host objects.

Service Status This represents the current status and attributes of Service objects.

Events These are typically time stamped messages that are generated by a monitoring system or managed device.

Host Alerts Generated when a host changes state.

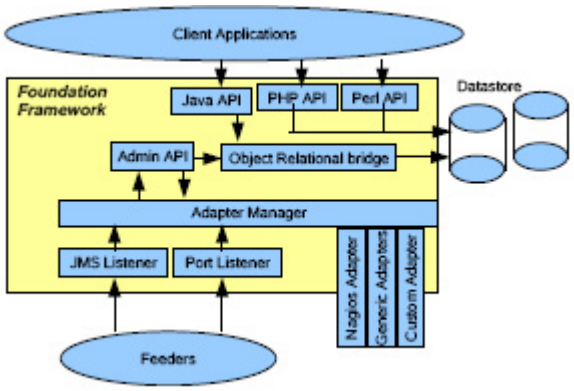
Host Notifications Generated when a notification occurs based on a Host Alert event.

Service Alerts Generated when a service changes state.

Service Notifications Generated when a notification occurs based on a Service Alert event.

The above description is taken from [43].Currently it has PHP, Perl and Java API's for retrieving its underling data. It has it own framework for receiving data as well. The foundation framework has flexible architecture to integrate data within its system as well as to retrieve form it. It will normalize the data so it can be retrieved in a consistent manner. The Foundation package includes Nagios, as the main monitoring system. The Web Service interface is a new addition to Foundation Framework.Following diagram 5.6 shows the different Foundation components and their interaction Ground-Work Open Source, Inc.

Figure 5.6: GWOS API architecture *source:http://gwfoundation.sourceforge.net/Foundation-bookshelf-2.0.1.pdf*



5.3.5 Implementation

Getting the data into the groundwork system to be integrated with its structure is quite flexible. One of the common method is to use their Groundwork Foundation platform. Another approach is to use their Nagios integration as a gateway to pass the data to its system. As groundwork uses Nagios as their core monitoring system , any information

gathered by a Nagios plugin can be integrated into the system. Nagios Feeders or the Event Broker in Groundworks take the information from the Nagios system and inserts it into its foundation database.

For this method, we have to create first the Host records and the services record for Nagios through groundwork. Services in Nagios falls into two categories

1. Active checks
2. Passive checks

Active checks in Nagios are scheduled in the Nagios schedule daemon where as passive checks are checks initiated by external process. Active checks are be used to poll a device or service for status information periodically. But there are cases when this scenario is not applicable such as when there are firewall involved or any other factors. Passive checks are useful for monitoring services that are asynchronous in nature and cannot be monitored effectively by polling their status on a regularly scheduled basis or services that cannot be effectively polled due to many restriction or in feasibility such as firewall restrictions.

All the service checks are performed by the plugins in Nagios. Plugins are compiled executables or scripts that can run from command line and report back the status of host or services. Nagios will process the results that it receives from the plugins and perform any necessary actions as configured for the service such as running event handlers, sending out notifications, etc.

For our implementation we will be using passive service check of Nagios to send data. Some details of how passive check works is described in coming section.

In passive service checks, external application or script checks the status of the host or the services records their data and status and writes the results of the check in Nagios external command file. Then Nagios reads this external command file and places the result of all passive checks into queue for later processing. Same queue are used for both active and passive checks so there are no differences between results of active and passive checks. This allows for seamless integration of status information from external applications with Nagios.

Figure below describes the general architecture of Nagios.

Submitting Passive Service Check Results External applications can submit passive service check results to Nagios by writing a PROCESS_SERVICE_CHECK_RESULT external command to the external command file. The format of the command is as follows:

```
[<timestamp>] PROCESS_SERVICE_CHECK_RESULT;<host_name>;<svc_description>;<return_code>;<plugin_output>
```

timestamp is the time in time_t format (seconds since the UNIX epoch) that the service check was performed (or submitted)

host_name is the short name of the host associated with the service in the service definition

svc_description is the description of the service as specified in the service definition

return_code is the return code of the check (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)

plugin_output is the text output of the service check (i.e. the plugin output)

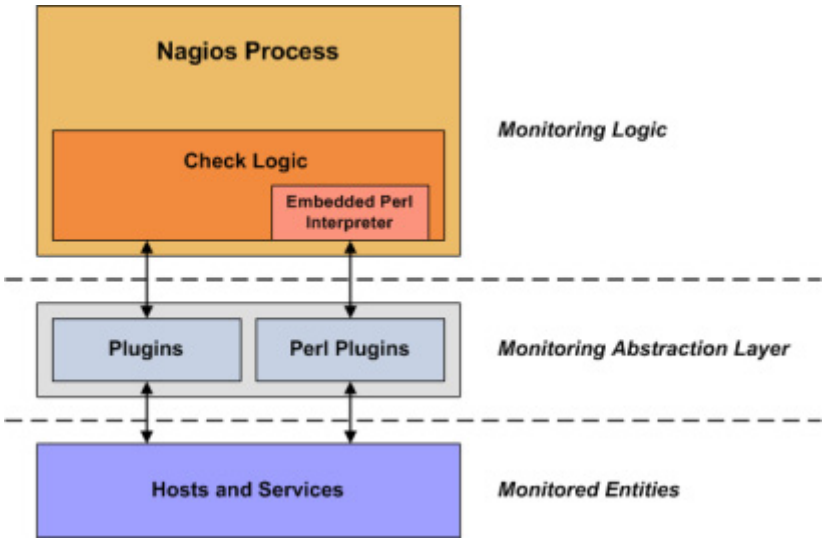


Figure 5.7: Nagios architecture *source:http://nagios.sourceforge.net/docs/3_0/plugins.html*

Submitting Passive Host Check Results External applications can submit passive service check results to Nagios by writing a PROCESS_HOST_CHECK_RESULT external command to the external command file. The format of the command is as follows:

```
[<timestamp>] PROCESS_HOST_CHECK_RESULT;<host_name>;<host_status>;<plugin_output>
```

timestamp is the time in time_t format (seconds since the UNIX epoch) that the service check was performed (or submitted)

host_name is the short name of the host associated with the service in the service definition

host_status is the status of the host (0=UP, 1=DOWN, 2=UNREACHABLE)

plugin_output is the text output of the host check

Submitting Passive Check Results From Remote Hosts If both nagios and the host are on same host then sending passive checks are easy by just writing the results in external command.But if the applications resides on different host than the nagios server then sending the checks results is a bit difficult. This is where a addon called NSCA comes into picture. The NSCA addon consists of a daemon that runs on the Nagios hosts and a client that is executed from remote hosts. The daemon will listen for connections from remote clients,as soon as it receives some check results it will validate it with the format described above and then write the check results into the external command file (as described above).

In groundworks NSCA daemon is already configured and listens on port 5667. So our implemented message formatter listen for any incoming messages either from CFEngine wrapper or Munin Wrapper, and as soon as it receives the message it converts the XML message to the format that NSCA addon expects and sends it. To send the message to NSCA in the remote host , a perl module called Net::NSCA::Client is used which can be found at `http://search.cpan.org/~dougdude/Net-`

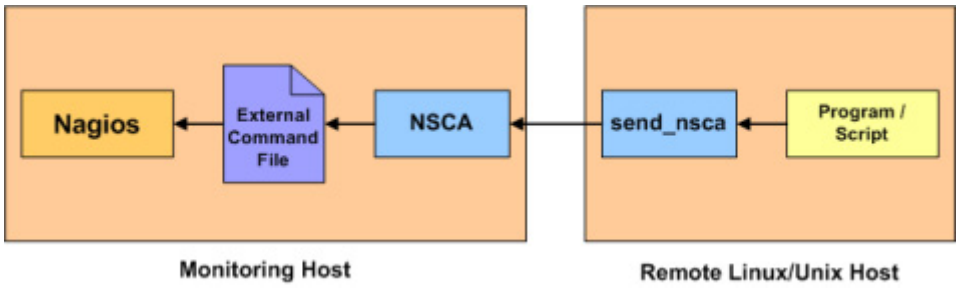


Figure 5.8: NSCA addon *source:http://nagios.sourceforge.net/docs/3_0/passivechecks.html*

`NSCA-Client-0.009002/lib/Net/NSCA/Client.pm`. Full implementation of the code is listed at A.4. Sample output of the message from the formatter is shown in listing 5.9

Listing 5.9: Formatter sending data to target system

```
At your service. Waiting...
sending to graphite....

CFEngine-cfengine-node-1.cfengine_out 0 1336260301

sending to nagios
cfengine-node-1 CFEngine-cfengine_out 0 OK OK, cfengine_out checked
    value is 0.|cfengine_out=0;0;0;0

At your service. Waiting...
sending to graphite....

CFEngine-cfengine-node-1.ssh_in 0 1336260301

sending to nagios
cfengine-node-1 CFEngine-ssh_in 0 OK OK, ssh_in checked value is 0.|
    ssh_in=0;0;0;0

At your service. Waiting...
sending to graphite....

CFEngine-cfengine-node-1.ssh_out 0 1336260301

sending to nagios
cfengine-node-1 CFEngine-ssh_out 0 OK OK, ssh_out checked value is
    0.|ssh_out=0;0;0;0

At your service. Waiting...
sending to graphite....

CFEngine-cfengine-node-1.cpu 0 1336260301

sending to nagios
cfengine-node-1 CFEngine-cpu 0 OK OK, cpu checked value is 0.|cpu
    =0;0;0;0

At your service. Waiting...
```

```
sending to graphite....

CFEngine-cfengine-node-1.syslog 0 1336260301
```

5.4 Data formatter for Graphite

Graphite is a scalable monitoring tool designed and written by Chris Davis at 2006. At 2008 it became a open source project under Apache 2.0 license. Graphites primary function is to store the time series data and then provides a interface where it render the data in flexible and scalable way. Graphite itself doesn't do any collection of data but utilizes other tools and depends on them to send data to it. So for this reason it has a very simple API to send data to it. Most common use case for using Graphite is to have a web based monitoring dash board for monitoring and analyzing the system state.

5.4.1 Architecture

Graphite is written entirely on python and consist of 3 software components

carbon a Twisted daemon that listens for time-series data

whisper - a simple database library for storing time-series data (similar in design to RRD)

graphite webapp - A Django webapp that renders graphs on-demand using Cairo

Fig 5.9 shows the general overview of the architecture of the Graphite.

5.4.2 Monitoring and Reporting

As graphite depends on external tools that collects and measures the performances , its up to users what they want to monitor and generate the graph for analyzing.

The Graphite has a simple URL-based API which can create custom graphs upon users request.A graphical interface built with javascript is also available for making this easy and simple.Figure 5.10 shows the graphing UI of the Graphite. Graphing of the matrices parameters are specified in the query-string of an HTTP GET request, the outcome of the processed query is returned in term of PNG image. For example, the URL: `http://graphiteserver/render?target=servers.www.cpuUsage&width=500&height=300` will requests a 500 x 300 graph for the metric servers.www.cpuUsage which is stored in the graphite system. There are extensive options available for manipulating the graphs. Graphite also has variety of available analytical functions that can be applied to the data once it is in the system.

5.4.3 Data storage model

Graphite uses whisper as the storage mechanism which is quite similar to round robin database used by rrdtool in design. It only store numeric time series data. whisper is a database library used by applications to manipulate and retrieve data stored in specially

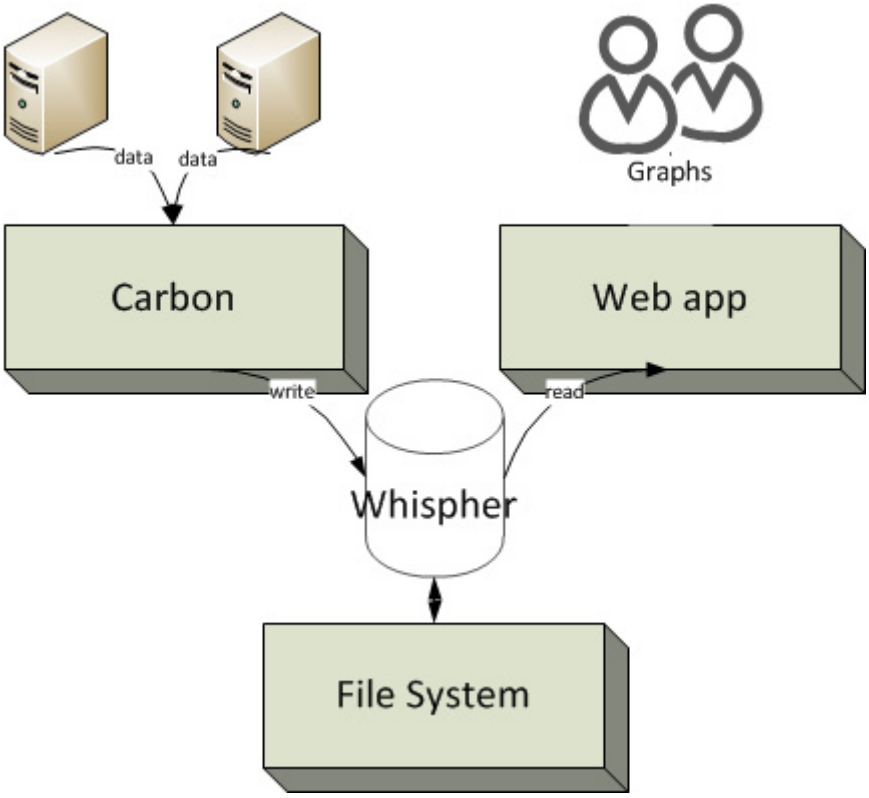


Figure 5.9: Grahite architecture

Figure 5.10: Graphite graph composer

formatted files. The most basic whisper operations are create to make a new whisper file, update to write new data points into a file, and fetch to retrieve data points.

Whisper files consist of a header section containing various metadata, followed by one or more archive sections . Each archive is a sequence of consecutive data points which are (timestamp, value) pairs. When an update or fetch operation is performed, whisper determines the offset in the file where data should be written to or read from, based on the timestamp and the archive configuration. Figure 5.11 shows general structure of the whisper storage model.

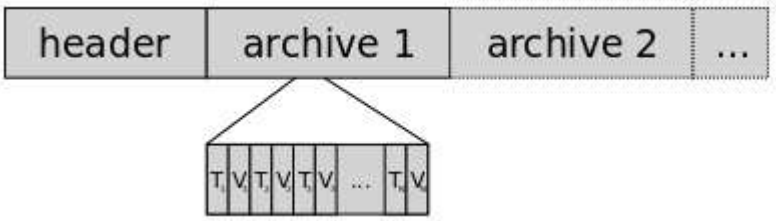


Figure 5.11: Whisper format

5.4.4 API

Graphite provides very easy API to get the measured matrices into the system. The most common use case in graphite is that there involves some monitoring agent which monitors the performance metrics and then sends it to graphite system through carbon. Metrics in Graphite have simple hierarchical names, similar to file system paths except that a dot is used to delimit the hierarchy rather than a slash or backslash. There are mainly two methods to get the data into graphite.

The plain text protocol The plain text protocol is the most straightforward protocol supported by Carbon. In this protocol when the monitoring agent sends data points to Graphite, First it must establish a TCP connection to carbon,default on port 2003. The client sends the metrics in a simple plain-text format. The format is one line of text per data point where each line contains the dotted metric name, value, and a Unix epoch timestamp separated by spaces. The data sent are in the format: ¡metric path¿ ¡metric value¿ ¡metric timestamp¿. Carbon will then transform the received message of text into a metric that the web interface and Whisper understand. Example munin.cpu.idle 88 123453234 can be sent through network socket where carbon is listening, by default the plain text protocol listen on port 2003. It generally works quite well if we want to send small amount of data quickly to graphite.

The pickle protocol The pickle protocol is a much more efficient take on the plain text protocol, and supports sending batches of metrics to Carbon in one go. The general idea is that the pickled data forms a list of multi-level tuples:[(path, (timestamp, value)), ...]

5.4.5 Implementation

The formatter was implemented using a perl script which would send graphite daemon a plain text message to port 2003.Its similar to the groundworks formatter but only the final format is changed to fit the graphite format.The detail implementation is listed at A.4 with similar output to 5.9.

Chapter 6

Results

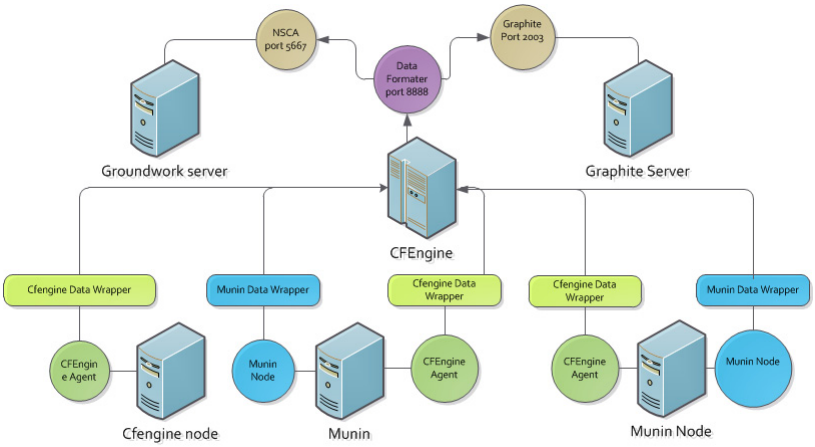


Figure 6.1: Test system architecture

The implementation was carried out in a small virtual infrastructure consisting of total of six machines of which two were running Munin instances and four of them were running CFEngine instances, one was groundwork server and one was graphite server. The Data Formatter daemon was kept running on one of the machine on port 8888, which would dispatch any incoming data that was sent from Munin data wrapper or CFEngine data wrapper.

For CFEngine, the DataWrapper is invoked by cfengine agents when they run in the machine. so data are reported by the agents and sent to data formatter through a perl script.

For Munin the Data Wrapper is invoked through a cron job setup to run the perl script every 5 minutes.

For CFengine only limited set of variables or metrics were measured which were important and was quite meaningful to measure. The metrics were

diskfree Free disk on / partition.

users Users with active processes.

rootprocs Sum of privileged system processes.

otherprocs Sum of non-privileged process.

loadavg Kernel load average utilization (sum over cores).

cfengine_in cfengine connections (in).

cfengine_out cfengine connections (out).

ssh_in ssh connections (in).

ssh_out ssh connections (out).

cpu %CPU utilization (all).

syslog New log entries (Syslog).

For Munin all the default plugins monitored value were integrated. The following metrics were defined

- disk
  - Disk IOs per device
  - Disk latency per device
  - Disk throughput per device
  - Disk usage in percent
  - Disk utilization per device
  - Inode usage in percent
  - IO Service time
  - IOstat
- munin
  - Munin processing time
- network
  - eth0 errors
  - eth0 traffic
  - Firewall Throughput
  - HTTP loadtime of a page
- processes
  - Fork rate
  - Number of threads
  - Processes
  - Processes priority
  - VMstat
- system
  - Available entropy
  - CPU usage
  - File table usage
  - Individual interrupts
  - Inode table usage

- Interrupts and context switches
- Load average
- Swap in/out
- Uptime

Both groundwork and graphite was setup and configured to receive the incoming data as described in implementation details.

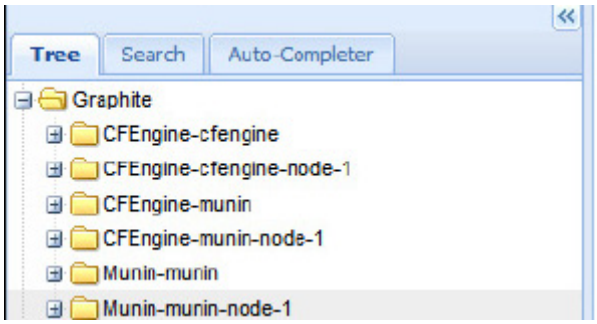


Figure 6.2: Graphite integration of machines

The figure 6.2 shows the integration of the results of the monitored machine in to the graphite web interface. The tree structure in the figure left shows the nodes that are being sending data to the graphite system. There are total 6 nodes that are shown in the figure of which four nodes are the the nodes that the CFEngine is monitoring. The nodes with the prefix “CFEngine“ are the nodes that are being monitored by the CFEngine agent and sending the data to graphite where as the nodes with the prefix “Munin” are referring to the nodes that are being monitored by Munin which in this case are two machines munin and munin-node-1.

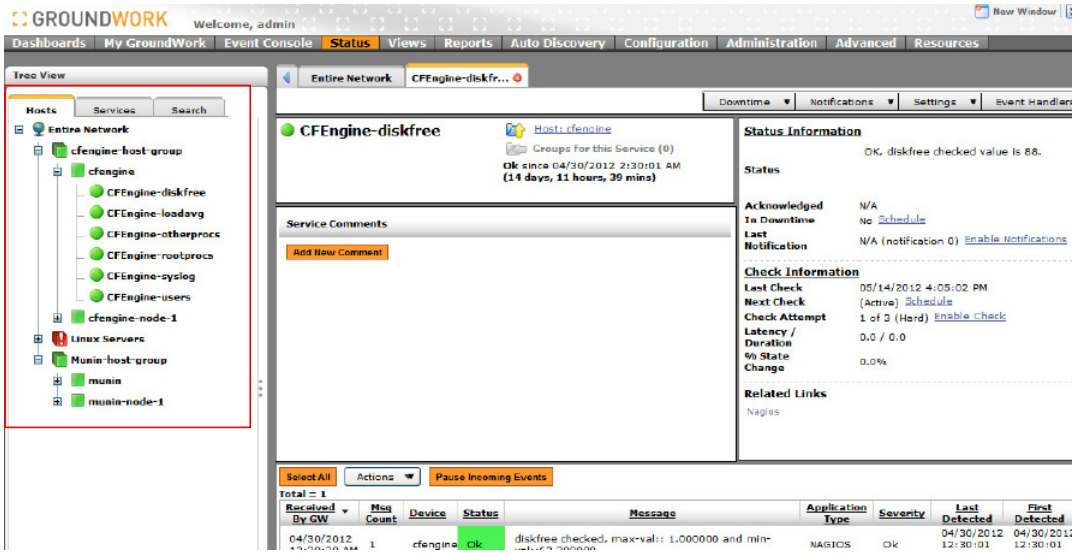


Figure 6.3: Groundwork integration result

The figure 6.3 shows the integration result in the groundwork web interface where the measured services and host are shown. The left side of the figure shows the nodes and the expanded nodes shows the services being monitored in that host. In this figure 4 nodes are shown grouped in two categories cfengine-host-group and munin-host-group which includes the host monitored by CFEngine and munin respectively. The expanded nodes shows the services that are being setup to collect data from monitored CFEngine services.

The next section presents some of the results that were obtained in the integration process. Although many of the services for both the CFEngine and Munin system were integrated, only some meaningful and interesting services are shown in this section. Some of the result of the matrices integrated into the system running on the implemented infrastructure are shown below.

6.1 Non-privileged process

The table 6.1 shows the data being collected by the CFEngine and then being reported over the period of three hours. The non-privileged process are the process stared by non-privileged users. The data in table are measured by cf-engine agent and in being reported to the data formatter in the interval of 5 minutes , i.e when the cf-agent runs on the system.

Table 6.1: Non privileged process measured by CFEngine over period of 3 hours

Time	cfengine	cfengine-node-1	munin	munin-node-1
5/11/12 1:00 PM	5	5	16	5
5/11/12 1:05 PM	5	5	17	5
5/11/12 1:10 PM	5	5	16	5
5/11/12 1:15 PM	5	5	17	5
5/11/12 1:20 PM	5	5	16	5
5/11/12 1:25 PM	5	5	17	5
5/11/12 1:30 PM	5	5	16	5
5/11/12 1:35 PM	5	5	16	5
5/11/12 1:40 PM	5	5	16	5
5/11/12 1:45 PM	5	5	16	5
5/11/12 1:50 PM	5	5	16	5
5/11/12 1:55 PM	5	5	16	5
5/11/12 2:00 PM	5	5	16	5
5/11/12 2:05 PM	5	5	16	5
5/11/12 2:10 PM	5	5	16	5
5/11/12 2:15 PM	5	5	16	5
5/11/12 2:20 PM	5	5	16	5
5/11/12 2:25 PM	5	5	16	5
5/11/12 2:30 PM	5	5	15	5
5/11/12 2:35 PM	5	5	15	5
5/11/12 2:41 PM	5	5	15	5
5/11/12 2:45 PM	5	5	15	5

Continued on next page

Table 6.1 – *Continued from previous page*

Time	cfengine	cfengine-node-1	munin	munin-node-1
5/11/12 2:50 PM	5	5	15	5
5/11/12 2:55 PM	5	5	15	5
5/11/12 3:00 PM	5	5	15	5
5/11/12 3:05 PM	5	5	15	5
5/11/12 3:10 PM	5	5	15	5
5/11/12 3:15 PM	5	5	15	5
5/11/12 3:20 PM	5	5	15	5
5/11/12 3:25 PM	5	5	15	5
5/11/12 3:30 PM	5	5	15	5
5/11/12 3:35 PM	5	5	16	5
5/11/12 3:40 PM	5	5	16	5
5/11/12 3:45 PM	5	5	16	5
5/11/12 3:50 PM	5	5	16	5
5/11/12 3:55 PM	5	5	16	6
5/11/12 4:00 PM	5	5	16	5

The information integrated into Graphite is shown as graph in 6.4.In this graph the monitored value of non-authorized users process is shown for all the four host that has been reporting the data over the time period of 5 hours. The x-axis defines the time and y-axis defines the value. The label are shown as the host names for where the "CFEngine" prefix is applied to the host that are monitored by the CFEngine indicating that these values are being integrated from CFEngine source.

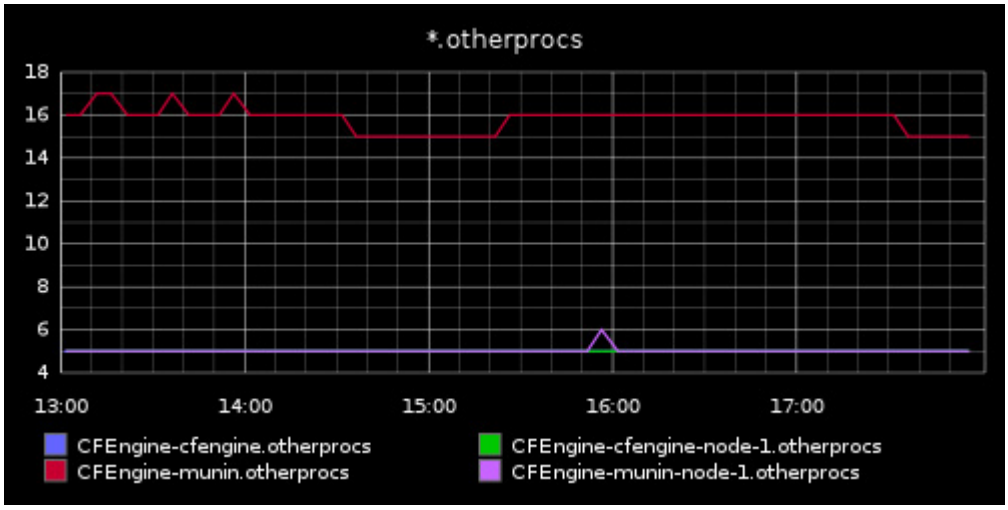


Figure 6.4: Graph in graphite for non previlaged process measured by cfengine

The same information integrated into Groundworks is shown as graph in 6.5. Here one of the features in groundwork to combine multiple performance view into one graph is being utilized to generate the graph. The data are integrated into groundwork system and stored which can be manipulated to have different graphs and reports about

the data. In current figure it shows the consolidated view of the four hosts measuring the non-privileged process reported by CFEngine. The four host are labeled in the legend with their respective name along with the service name being reported. Here also the x-axis shows the time interval and the y-axis shows the value of the measured service.

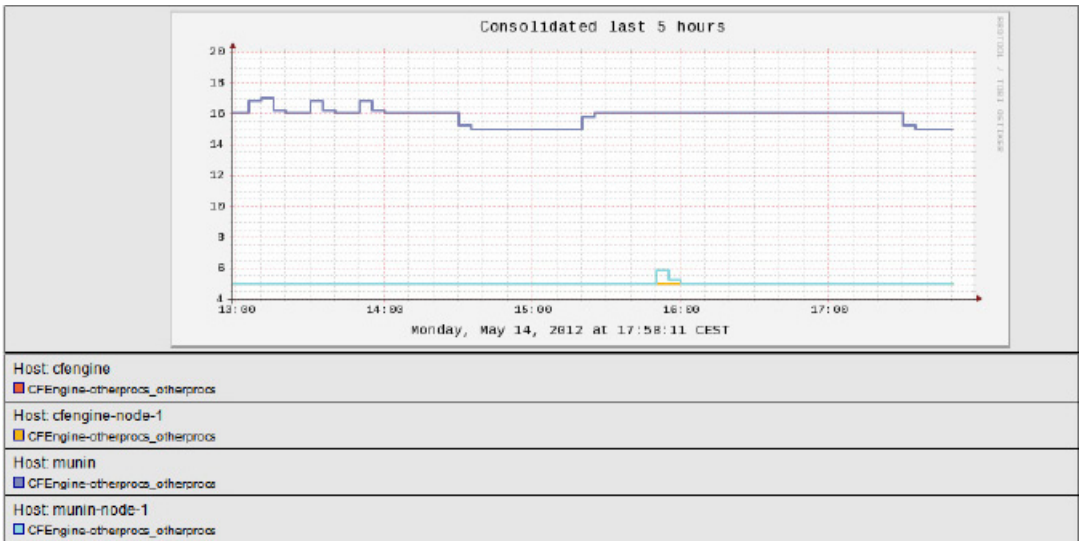


Figure 6.5: Graph in graphite for non privileged process measured by cfengine

6.2 Munin process threads

Table 6.2 shows the process threads value collected by munin on two munin node over the period of time. Generally threads are light weight process, which generally share resource while being contained inside a specific process. The table shows the measured value as reported by the munin system running on two nodes. The values shown were reported by munin system on over interval of 3 hours which was integrated in both graphite and groundwork system.

Table 6.2: Process threads measured by Munin on two machines

Time	munin	munin-node-1
5/11/12 12:02 PM	86	81
5/11/12 12:07 PM	90	84
5/11/12 12:07 PM	93	84
5/11/12 12:12 PM	93	77
5/11/12 12:17 PM	86	77
5/11/12 12:22 PM	86	77
5/11/12 12:27 PM	86	77
5/11/12 12:32 PM	86	77
5/11/12 12:37 PM	86	77

Continued on next page

Table 6.2 – *Continued from previous page*

Time	munin	munin-node-1
5/11/12 12:42 PM	86	77
5/11/12 12:47 PM	86	77
5/11/12 12:52 PM	86	77
5/11/12 12:57 PM	86	77
5/11/12 1:02 PM	86	81
5/11/12 1:07 PM	90	81
5/11/12 1:07 PM	93	81
5/11/12 1:12 PM	93	77
5/11/12 1:17 PM	86	77
5/11/12 1:22 PM	86	77
5/11/12 1:27 PM	86	77
5/11/12 1:32 PM	86	77
5/11/12 1:37 PM	86	77
5/11/12 1:42 PM	86	77
5/11/12 1:47 PM	86	77
5/11/12 1:52 PM	86	77
5/11/12 1:57 PM	86	77
5/11/12 2:02 PM	86	81
5/11/12 2:07 PM	90	80
5/11/12 2:07 PM	93	81
5/11/12 2:12 PM	93	77
5/11/12 2:17 PM	86	77
5/11/12 2:22 PM	86	77
5/11/12 2:27 PM	86	77
5/11/12 2:32 PM	86	77
5/11/12 2:37 PM	86	77
5/11/12 2:42 PM	86	77
5/11/12 2:47 PM	86	77
5/11/12 2:52 PM	86	77
5/11/12 2:57 PM	86	77
5/11/12 3:02 PM	86	81
5/11/12 3:07 PM	90	84

Graphite integration The graph 6.6 shows the munin data integrated into the graphite system and then represented in graphite web interface. As in above examples the data's are shown in a single graph for both the munin-node. The graphs shows two munin host as labeled munin and munin-node-1 along with the prefix "Munin" to indicate the data source as Munin and also the metric category process separated by "." symbol. The x-axis shows the time period of the measured value and y-axis shows the values being measured by the munin.

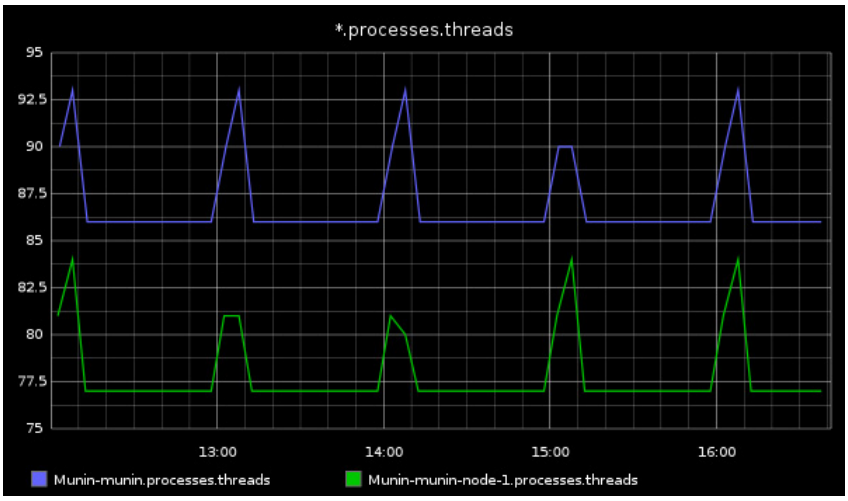


Figure 6.6: Graphite graph of process threads measured by munin

Groundwork integration The graph 6.7 shows the same information integrated in the groundwork and then represented in groundwork web portal information. The graph is generated by consolidating the two host and the process threads data reported by munin into the groundwork system. The graph here indicates the two hosts in the label and with the service description. The x-axis indicates the graphs time periods and y-value indicates the process threads number.

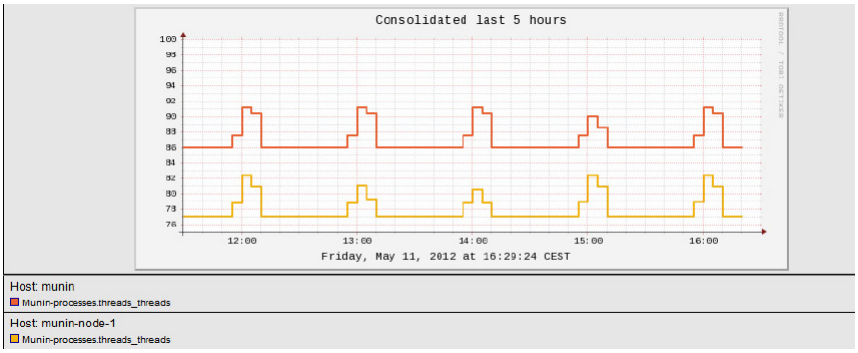


Figure 6.7: Groundworks graphs of the thread process information

6.3 CFEngine Root process

Table 6.3 shows the values reported by CFEngine for the four machines which was integrated with groundworks and graphite. The root process is the measure of sum of privileged process running on the system. The table shows the value being measured by CFEngine for the root process running on the system in the interval of 5 min and then reporting it for the integration.

Table 6.3: CFEngine root process measurement on 4 machine

Time	CFEngine	CFEngine-node-1	munin	munin-node-1
5/12/12 12:00 AM	68	58	61	63
5/12/12 12:05 AM	68	58	65	67
5/12/12 12:10 AM	68	58	61	63
5/12/12 12:15 AM	68	58	61	63
5/12/12 12:20 AM	68	58	61	63
5/12/12 12:25 AM	68	58	61	63
5/12/12 12:30 AM	68	58	61	63
5/12/12 12:35 AM	68	58	61	63
5/12/12 12:40 AM	68	58	61	63
5/12/12 12:45 AM	68	58	61	63
5/12/12 12:50 AM	68	58	61	63
5/12/12 12:55 AM	68	58	61	63
5/12/12 1:00 AM	68	58	61	63
5/12/12 1:05 AM	68	58	65	67
5/12/12 1:10 AM	68	58	62	63
5/12/12 1:15 AM	68	58	62	63
5/12/12 1:20 AM	68	58	62	63
5/12/12 1:25 AM	68	58	62	63
5/12/12 1:30 AM	68	58	62	63
5/12/12 1:35 AM	68	58	62	63
5/12/12 1:40 AM	68	58	62	63
5/12/12 1:45 AM	68	58	62	63
5/12/12 1:50 AM	68	58	62	63
5/12/12 1:55 AM	68	58	62	63
5/12/12 2:00 AM	68	58	62	63
5/12/12 2:05 AM	68	58	66	67
5/12/12 2:10 AM	68	58	62	63
5/12/12 2:15 AM	68	58	62	63
5/12/12 2:20 AM	68	58	62	63
5/12/12 2:25 AM	68	58	62	63
5/12/12 2:30 AM	68	58	62	63
5/12/12 2:35 AM	68	58	62	63
5/12/12 2:40 AM	68	58	62	63
5/12/12 2:45 AM	68	58	62	63
5/12/12 2:50 AM	68	58	61	63
5/12/12 2:55 AM	68	58	62	63
5/12/12 3:00 AM	68	58	61	63
5/12/12 3:05 AM	68	58	65	67
5/12/12 3:10 AM	68	58	61	63
5/12/12 3:15 AM	68	58	61	63
5/12/12 3:20 AM	68	58	61	63
5/12/12 3:25 AM	68	58	61	63
5/12/12 3:30 AM	68	58	61	63

Continued on next page

Table 6.3 – *Continued from previous page*

Time	CFEngine	CFEngine-node-1	munin	munin-node-1
5/12/12 3:35 AM	68	58	61	63
5/12/12 3:40 AM	68	58	61	63
5/12/12 3:45 AM	68	58	61	63
5/12/12 3:50 AM	68	58	61	63
5/12/12 3:55 AM	68	58	61	63
5/12/12 4:00 AM	68	58	61	63

Figure 6.8 shows the graphical representation of the above information in graphite web interface. The graph is generated in graphite interface with composite information from all the four host to show the root process running in different nodes running CFEngine. As seen on the graph the nodes are labeled with CFEngine as prefix to indicate the source of this data is coming from CFEngine agent.And the label is appended with rootprocs which is the service name measuring the root process in the system. The x-axis here denotes the time periods of the measured value where the y-axis denotes the value of the root processes running in different nodes.

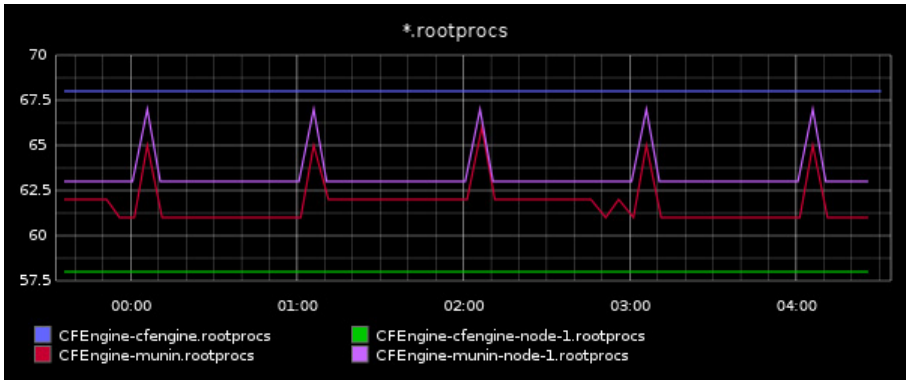


Figure 6.8: Graphite representation of root process

6.4 Munin system load average

The load average of the machine describes how many processes are in the run-queue (scheduled to run ”immediately”).This metric is measured by munin on the two nodes that are running munin system , munin and munin-node-1. The data is shown on the table 6.4 which shows the munin measurement of the system load average value over the period of time. The information is then dispatched to the groundwork and graphite system for integration.

Table 6.4: Munin load

Time	munin	munin-node-1
5/12/12 12:00 AM	0.24	0.06
5/12/12 12:05 AM	0.22	0.04
5/12/12 12:10 AM	0.22	0.04
5/12/12 12:15 AM	0.25	0.06
5/12/12 12:20 AM	0.22	0.06
5/12/12 12:25 AM	0.21	0.07
5/12/12 12:30 AM	0.15	0.04
5/12/12 12:35 AM	0.13	0.04
5/12/12 12:40 AM	0.11	0.05
5/12/12 12:45 AM	0.1	0.22
5/12/12 12:50 AM	0.17	0.15
5/12/12 12:55 AM	0.2	0.23
5/12/12 1:00 AM	0.24	0.18
5/12/12 1:05 AM	0.22	0.3
5/12/12 1:05 AM	0.28	0.3
5/12/12 1:10 AM	0.28	0.2
5/12/12 1:15 AM	0.26	0.18
5/12/12 1:20 AM	0.22	0.15
5/12/12 1:25 AM	0.28	0.06
5/12/12 1:30 AM	0.29	0.14
5/12/12 1:35 AM	0.29	0.07
5/12/12 1:40 AM	0.23	0.07
5/12/12 1:45 AM	0.29	0.08
5/12/12 1:50 AM	0.26	0.06
5/12/12 1:55 AM	0.24	0.03
5/12/12 2:00 AM	0.23	0.02
5/12/12 2:05 AM	0.23	0.05
5/12/12 2:05 AM	0.15	0.05
5/12/12 2:10 AM	0.15	0.03
5/12/12 2:15 AM	0.27	0.01
5/12/12 2:20 AM	0.38	0.13
5/12/12 2:25 AM	0.21	0.16
5/12/12 2:30 AM	0.27	0.19
5/12/12 2:35 AM	0.15	0.27
5/12/12 2:40 AM	0.18	0.26
5/12/12 2:45 AM	0.25	0.11
5/12/12 2:50 AM	0.3	0.24
5/12/12 2:55 AM	0.31	0.11
5/12/12 3:00 AM	0.19	0.2
5/12/12 3:05 AM	0.27	0.26
5/12/12 3:05 AM	0.37	0.26
5/12/12 3:10 AM	0.37	0.11
5/12/12 3:15 AM	0.23	0.05

Continued on next page

Table 6.4 – *Continued from previous page*

Time	munin	munin-node-1
5/12/12 3:20 AM	0.28	0.09
5/12/12 3:25 AM	0.35	0.06
5/12/12 3:30 AM	0.26	0.04
5/12/12 3:35 AM	0.3	0.02
5/12/12 3:40 AM	0.24	0.01
5/12/12 3:45 AM	0.19	0.01
5/12/12 3:50 AM	0.25	0.05
5/12/12 3:55 AM	0.32	0.03
5/12/12 4:00 AM	0.18	0.04
5/12/12 4:05 AM	0.12	0.02

The figure 6.9 shows how the information integrated on the graphite system is being displayed and analyzed. The graph shows the data from munin source for system load average values reported by munin on two nodes running munin. The graph is consolidated by host show that the values can be seen on one graph. The labels shows the name of the host with the prefix "Munin" to indicated the data source along with the name of the service i.e "system.load". The x-axis is the time period of the measured value and y axis indicates the system load values measured.

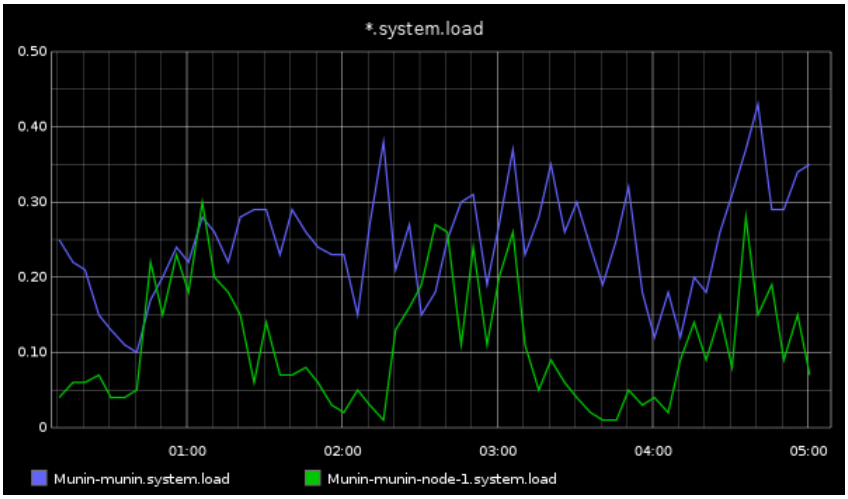


Figure 6.9: Munin sytem load of two machines

The figure 6.10 indicates the same information represented by the groundwork web interface for generating reports. The graphs shows the consolidate view of two munin hosts showing the system load average values over the period of 5 hours. The graph indicates the values measured for system load average for host called munin on top and then at the bottom then graph is for system load average for the host "munin-node-1".

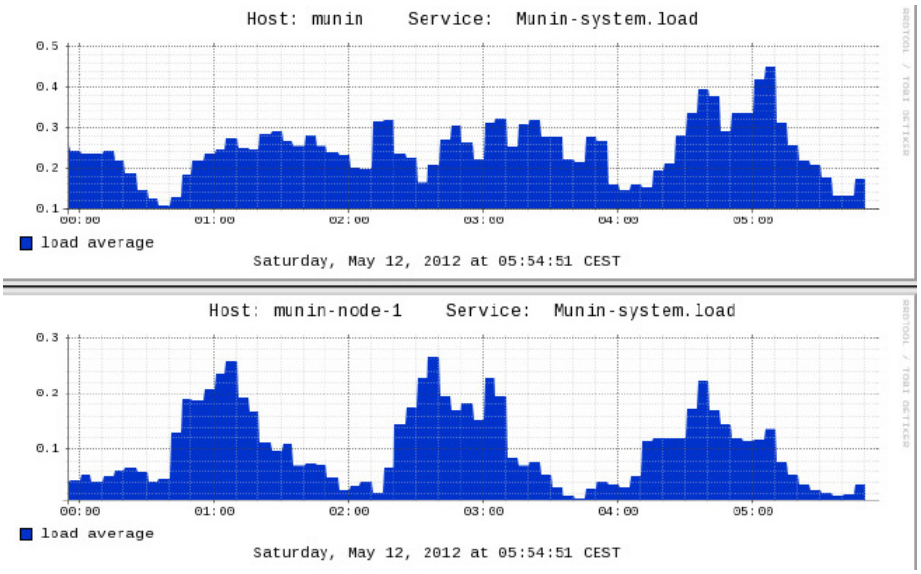


Figure 6.10: Munin system load from groundworks

6.5 Disk Free

Disk free service is the measurement of the free disk space in "/" partition calculated in percentage value by CFEngine. For the disk free service, testing for the warning in disk free service was tested by filling up one of the node machine (cfeninge-node-1) with huge 13GB file on total of 15GB of hard disk by running the following command

```
dd if=/dev/zero of=filename bs=$((1024*1024)) count=$((13*1024))
```

The following figure 6.11 shows the warning in groundwork web interface where the diskfree service is labeled as critical. As seen in the figure in the left region, the highlighted service has been marked as critical by groundworks by having a red icon of exclamation on the side of the service instead of the green circle status indicating the OK status of the service. Also the right side of the figure shows the value as 5% that is measured by the CFEngine agent running on the node. The picture also shows how the host icon is also labeled critical as one of the services go critical on the nodes.

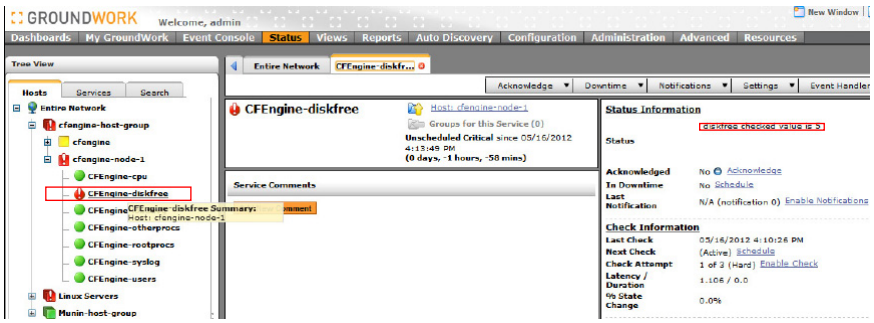


Figure 6.11: Diskfree service critical in groundworks

The same information can also be seen on graphite in the following graph 6.12 showing the critical change in the disk free service. The graph below shows the information of the disk free values measured by the CFEngine agent over the period of the time. we can see how the disk percent of the system is constant over the period of time then suddenly drops to below 5% as indicated by the greeen line. The lengend indicates that the green line is for the host cfengine-node-1 which is prefixed by “CFEngine” to indicate the source of the data.The x axis defines the percentage of disk free on the system and the y-axis indicates the time period over which the values are measured.

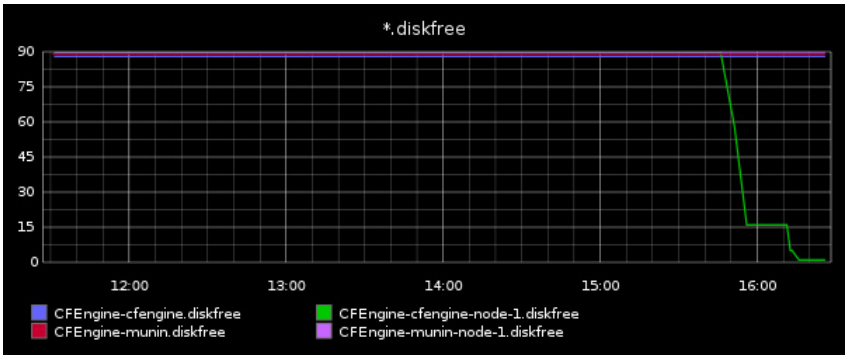


Figure 6.12: Graphite diskfree service

6.6 Syslog

The syslog is the measure of new system log over the interval of time on any given system. This section shows the warning generated by CFEngine agents when the high number of syslog is detected than usual trend which is then reflected in the groundwork web interface as well as in graphite system. The values and state of the syslog service is constantly updated by CFEngine agent and then reported for data integration.

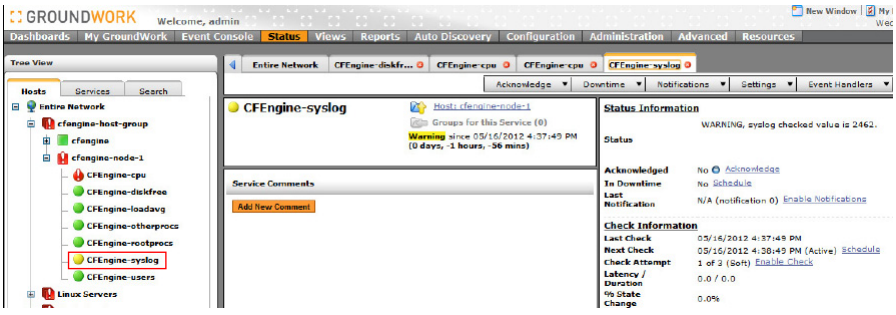


Figure 6.13: Groundwork syslog

The figure 6.13 showing the data reported by CFEngine agents about the warning state of the syslog entries. Here in the figure the highlighted area by red triangle shows the warning state of the syslog service in the host named cfengine-node-I. The warning state is indicated by yellow round circle along side of the service.

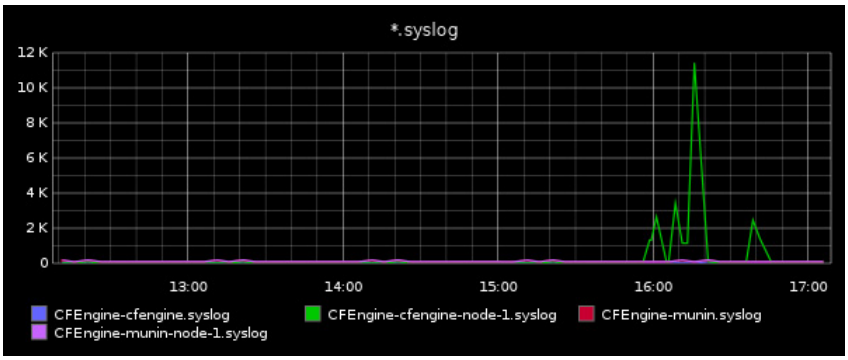


Figure 6.14: Graphite syslog graph measured by CFEngine

The figure 6.14 shows the graphite representation of the syslog values reported by the CFEngine agents for four host that has been monitored by CFEngine and has been reporting the data for integration. As seen in the figure the syslog entries are dramatically increased between the period of around 15.00 which clearly shows the break in trend form the past values reported. The graph shows the nodes name in the label with “CFEngine” as prefix to indicate the cfengine source. The x-axis shows the time period of the reported value and the y-axis indicated the value of new entries in syslog. Here the green line shows the generated warning syslog.

Chapter 7

Discussion and future work

In todays monitoring scenario the widely used tools are available with different features and capabilities. The infrastructures are growing complex and heterogeneous both in terms of platforms and machines used, and also in the tools that are being used to manage and monitor those platforms. The data these monitoring tools measures are stored in different systems, differ in formats and has different semantics. Having diverse tools gives rise to some problems that are described in the chapter 3. One of the way to tackle or handle those problems is through data integration of these different heterogeneous system.

Basically when dealing with data integration problem, heterogeneity of the data sources needs to be tackled. Each of the data sources may be implemented on different platforms such as OS, database, file systems etc. Also the sources are based on different conceptual models which leads to semantic heterogeneity between different sources. Semantic heterogeneity refers as modeling of same data information with different names or can be referred as difference in conceptual representation of data. The system level heterogeneity is nowadays quite easy to overcome with the availability of many tools and technologies but to overcome the semantic difference between different source is not an easy task. The resolution of the semantic differences in data integration has been researched [44] ,[45],[46] for quite some time, which would be a key step in defining a common data model for the data integration.

7.1 Tools selection for integration

The monitoring tools in this thesis that were chosen for the prototype implementation was because of their wide usage and popularity on current monitoring scenario. As there were many other tools than the ones used in this prototype, they were not included due to the limited time that was available to study the internals and the specification of the tools. Also the selected tools chosen were highly different in architecture, philosophy , features and inner workings. Those tools represented most of the varied solution deployed on the monitoring scenario nowadays.

7.2 Implementation

The proof of concept implementation of the common data model in this thesis only covers the domain for small numbers of tools that was selected for the implementation. It does not tries to come up with the complete full featured common model, but rather tries to prove that having a common model similar to the one implemented and having a full standardized schema for the common model to be implemented by different tools can help mitigate some complexities managing those tools. This approach of solving the problem requires the overhead of coming up with suitable ontologies for covering up each data source.An ontology is an explicit specification of a conceptualization[47].

Existing approaches on data sharing and tools integration that relies on implementing wrapper libraries for directly converting data between different formats, such as seen on the implementation, have several limitations. Implementing and maintaining wrapper layer for different sources requires high implementation and maintenance costs. And as the sources of data increases or changes the maintainability of those thing also becomes problematic.

Also the semantic of data is not always maintained while converting data's between the sources through the wrappers. Although simple XML schema's such as the prototype implemented can be useful in certain scenarios they must be sharable and must establish a common understanding about the data, which would maintain semantic interoperability between various parties which is an important issue that monitoring and measurement tools have to support.

As seen from the results in the previous chapter, we can see that a simple common data model with a common global semantic can be a very useful and efficient factor for building a co-operative environment between heterogeneous tools where different tools specialize in their own domain like data collection, data analysis, visualization etc. By introducing a standard common model which is both rich in context and representation can help solve difficulties in data exchange between different sources which in turn can help solve different problems that the current monitoring scenarios is facing ,such as the common interface system where we can see the aggregated data in one single place instead of having to look for the information in different places. It can also solve the visualization problem that are inefficient or limited in capabilities by exchanging the data with more specialized tool that has been designed specially for the visualization purpose.

As the results show that the data monitored from CFEngine and Munin are integrated in the Graphite system, it allows enhanced graphing capability of the same information along with high degree of other other data manipulating functionality such as combining, comparing, manipulating, exporting etc which are lacking or very limited in both of the tools. The integration helps the users better understand the data and have better visualization of the data leading in more efficient and useful monitoring experience.

Similarly the integration into the groundwork system of the monitoring data's of CFEngine and Munin provides a single interface to monitor the different heterogeneous system, such as systems monitored by CFEngine as well as systems that are monitored by Munin. The benefit of having a single interface for these different system is that the time needed for looking up the information or monitoring is vastly reduced because the user doesn't have to shuffle between different interfaces to get

the information.The consolidation of information in one place with consistent interface also resolves the ambiguity or confusion that otherwise two different interface may create. As research has stated that Inconsistent interface terminology slowed user performance by 10 to 25 percent[48]. It is also proved that the interface consistency shortens the learning process , reduced working memory demand and increased efficiency. [49]. The consolidation of data also helps the users to tackle the monitoring administration of maintaining large system treating different part of infrastructure as a single unit.

7.3 Problems

Some of the problems encountered while implementing the prototype was the lack of proper documents for the working of the tools and its API's. There were documents available but not always updated or without the sufficient information from the perspective of data integration .The wrapper and formatter layer could have been more robust and fault tolerance, the current implementation doesn't have robust error and validation in them. The lack of adequate hardware for testing also limited the scale of infrastructure to test upon. Also one of the hard part of coming up with the prototype model was the common XML format due to the previously discussed semantic heterogeneity between the selected tools. Example CFEngine, Munin, Groundworks all have different concept of service states. In CFEngine it treats the service state in terms of anomalies and deviations while munin does not have concrete definition of service states where as in Groundwork it follows the Nagios convention of critical, Warning, OK etc.

7.4 Future work

Many things on this implementation can be implemented and tested more robustly, but for future the most important thing would be to have a standardized ontologies implemented between the tools which would be a core basis for data exchange and integration between heterogeneous tools. A more concrete way would be to adopt already researched and existing prototype or schema to use instead of coming up with custom solution, such as use a unified information model like CIM. Another thing that can be done is to modify the open source tools to natively support these common model processing instead of creating a wrapper around them.Also it would be interesting to study about the implementation of transforming the tools to adopt the common model.

Chapter 8

Conclusion

Network monitoring tools are a vital part of network operation these days.With availability of wide ranges of choices of such tools,it increase the heterogeneity in the infrastructure that arises many problems. The data exchange between the different tools becomes much more complex with the increasing heterogeneity between the tools. This report and implementation describes the benefits of having a standardized data format between the tools. New open standards and the adoption of common data model to formally define and represent data knowledge, are the main features of these architectures that enable the definition of cooperative environments. The implementation also shows that if all tools agrees on a standard common information model and expose their software to be able to process the model, then managing the heterogeneous environment of tools would be less cumbersome and more efficient. Designing a standard data integration framework or system is a complex task which requires great deal of expertise and resources , but has its benefits and certainly the need is also there. Fortunately, a lot of research are being carried out in this topic which are mentioned on the discussion and background chapters.Lots of standard formats are also being purposed which are evolving and may be in near future all the tools will support one common data model to have a manageable heterogeneous environment.

Appendix A

Source code

A.1 CFEngine Data Wrapper

Listing A.1: CFEngine data wrapper perl script

```
#!/usr/bin/perl

# Needed packages
use Getopt::Std;
use strict "vars"; # declare variable
use Data::Dumper;
use IO::Socket;
use XML::Simple;

# Global variable
my $VERBOSE = 0 ;
my $DEBUG = 0;

# Handle flags and arguments

# Example :: c == "-c", c: == "-c argument"
my $optString = 'vdh1:2:c:';
getopts($optString,\my %opt) or usage() and exit 1;

# Print  help message in -h is invoked


if ( $opt{'h'} ) {
    usage();
    exit 0;
}




$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
```

```
verbose("Verbose is enabled");
debug("Debug is enabled");



# Main


my $FORMATTER_IP = "10.0.0.6";
my $FORMATTER_PORT = "8888";


# run the program and capture the output
debug("Running command :: /var/cfengine/bin/cf-report -vI");
my $output = qx (/var/cfengine/bin/cf-report -vI);
# print $output;
# get host name from the output
my $hostname = "";
if ($output =~ m/Host name is:\s+(.*)/) {
  $hostname = $1;
}

## get ip


my $IP_eth0 = qx (/sbin/ifconfig eth0);
$IP_eth0 =~ s/.*inet addr:(.*)  Bcast:/1/;
my $ip = $1;
print "IP eth0 => " . $ip . "\n";

# print $hostName;

my $FILEPATH = "/var/cfengine/reports/monitor_summary.csv";
my $CLASSPATH = "/var/cfengine/reports/classes.csv";
my $xml = new XML::Simple();

my @serviceArray;


my $dataHash = (
    'Host' => {
        'time' => time,
        'status'  => 'OK',
        'ip' => $ip,
        'name' => $hostname,
        'source' => 'CFEngine',
        'service' =>\@serviceArray
    }
);



open(CSV,$FILEPATH);
while (my $line = <CSV>) {

my @metrics = split (/,/, $line);
```

```
    my $metric = trim($metrics[1]);
    my $minValue = trim($metrics[2]);
    my $maxValue = trim($metrics[3]);
    my $sigma = trim($metrics[4]);

     my $now = time ;

     my $state =  checkForWarningClasses($metric);
     my $message = "$metric checked, max-val:: $maxValue and min-val:: $minValue";
     my %service = (
            'Max', '',
            'Unit', 'float',
            'Warn', '',
            'MeasuredTime', $now,
            'State', $state,
            'Message', $message,
            'Min' , '',
            'Label', '',
            'Critical', '',
            'MeasuredValue' , $minValue,
            'Type', $metric
            );

     push @serviceArray,\%service;



}


close(CSV);


my $xmlmessage = createServiceXml();
debug("XML Format created ::");
debug($xmlmessage);

sendToFormatter($xmlmessage);

exit(0);



sub sendToFormatter(){

 debug("sending message to formatter");
 my $message = $_[0];

 my $gSocket = new IO::Socket::INET (   PeerAddr => $FORMATTER_IP,
                                        PeerPort => $FORMATTER_PORT,
                                        Proto => 'tcp',);
 die "Could not create formatter socket : $!\n" unless $gSocket;
 $gSocket->autoflush(1);
 debug($message);
 print $gSocket $message;
 close($gSocket);
```

```
}


# returns xml from the hash
sub createServiceXml() {

    return $xml->XMLout($dataHash,RootName=>'Host',NoAttr=>1,KeyAttr => []);

}




sub checkForWarningClasses() {

    my $class = $_[0];
    my $state = 'OK'; # default state
    ## look for warning classes with _dev2 classes set
    open (CLASSES,$CLASSPATH);
    while (my $classLine = <CLASSES>) {

        my $warningRegex = $class."_(.*)_dev2";
        my $criticalRegex = $class."_(.*)_anomaly";
        if ($classLine =~ m/$warningRegex/) {
            $state = "warning";
        }elsif ($classLine =~ m/$criticalRegex/) {
            $state = "critical";
        }
    }

    return $state;

}


# Perl trim function to remove whitespace from the start and end of the string
sub trim($)
{
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

sub usage {
    print "Usage: \n";
    print "-h Usage \n";
    print "-v Verbose \n";
    print "-d Debug \n";
    print "-1 input file one \n";
    print "-2 input file two \n";
    print "-c column number \n";
    print "./script [-d] [-v] [-h] -1 <file1> -2 <file2> -c <columnNumber> \n";
}
```

```
sub verbose {
    print $_[0]."\n" if $VERBOSE;
}

sub debug {
    print $_[0]."\n" if $DEBUG;
}
```

Listing A.2: CFEngine data wrapper for second approach perl script

```
#!/usr/bin/perl

# Needed packages
use Getopt::Std;
use strict "vars"; # declare variable
use Data::Dumper;
use IO::Socket;
use XML::Simple;

# Global variable
my $VERBOSE = 0 ;
my $DEBUG = 0;

# Handle flags and arguments

# Example :: c == "-c", c: == "-c argument"
my $optString = 'vdhi:n:s:c:m:';
getopts($optString,\my %opt) or usage() and exit 1;

# Print  help message in -h is invoked


if ( $opt{'h'} ) {
    usage();
    exit 0;
}




$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};



verbose("Verbose is enabled");
debug("Debug is enabled");



# Main


my $FORMATTER_IP = "10.0.0.6";
my $FORMATTER_PORT = "8888";
```

```
my $ip = $opt{'i'};
my $hostname  = $opt{'n'};
my $state = $opt{'s'};
my $metric = $opt{'m'};
my $value = $opt{'c'};
my $date = time;


my $FILEPATH = "/var/cfengine/reports/monitor_summary.csv";
my $CLASSPATH = "/var/cfengine/reports/classes.csv";
my $xml = new XML::Simple();

my @serviceArray;


my $dataHash = (
    'Host' => {
        'time' => time,
        'status'  => 'OK',
        'ip' => $ip,
        'name' => $hostname,
        'source' => 'CFEngine',
        'service' =>\@serviceArray
    }
);


    my $message = "$state, $metric checked value is $value.|$metric=$value;0;0;0";
    my %service = (
          'Max', '',
          'Unit', 'float',
          'Warn', '',
          'MeasuredTime', $date,
          'State', $state,
          'Message', $message,
          'Min' , '',
          'Label', '',
          'Critical', '',
          'MeasuredValue' , $value,
          'Type', $metric
          );

    push @serviceArray,\%service;



my $xmlmessage = createServiceXml();
debug("XML Format created ::");
debug($xmlmessage);

sendToFormatter($xmlmessage);

exit(0);
```

```
sub sendToFormatter(){

 debug("sending message to formatter");
 my $message = $_[0];

 my $gSocket = new IO::Socket::INET (   PeerAddr => $FORMATTER_IP,
                                        PeerPort => $FORMATTER_PORT,
                                        Proto => 'tcp',);
 die "Could not create formatter socket : $!\n" unless $gSocket;
 $gSocket->autoflush(1);
 debug($message);
 print $gSocket $message;
 close($gSocket);


}


# returns xml from the hash
sub createServiceXml() {

    return $xml->XMLout($dataHash,RootName=>'Host',NoAttr=>1,KeyAttr => []);

}




# Perl trim function to remove whitespace from the start and end of the string
sub trim($)
{
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

sub usage {
    print "Usage: \n";
    print "-h Usage \n";
    print "-v Verbose \n";
    print "-d Debug \n";
    print "-1 input file one \n";
    print "-2 input file two \n";
    print "-c column number \n";
    print "./script [-d] [-v] [-h] -i ip -h hostname -m metric -v value -s status \n";
}

sub verbose {
    print $_[0]."\n" if $VERBOSE;
}

sub debug {
```

```
    print $_[0]."\n" if $DEBUG;
}
```

A.2 Munin data formatter

Listing A.3: Data wrapper script for munin

```
#!/usr/bin/perl

# Needed packages
use Getopt::Std;
use strict "vars"; # declare variable
use Data::Dumper;
use IO::Socket;
use XML::Simple;

# Global variable
my $VERBOSE = 0 ;
my $DEBUG = 0;

# Handle flags and arguments

# Example :: c == "-c", c: == "-c argument"
my $optString = 'vdh1:2:c:';
getopts($optString,\my %opt) or usage() and exit 1;

# Print  help message in -h is invoked


if ( $opt{'h'} ) {
        usage();
        exit 0;
    }




$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};



verbose("Verbose is enabled");
debug("Debug is enabled");



# Main

my $muninSocket;
my $xml = new XML::Simple();

my @serviceArray;

my $IP_eth0 = qx (ifconfig eth0);
$IP_eth0 =~ s/.*inet addr:(.*)  Bcast:/1/;
```

```
my $ip = $1;
print "IP eth0 => " . $ip . "\n";


my $host =(grep {chomp;} system ('hostname -s'))[0];


my $FORMATTER_IP = "10.0.0.6";
my $FORMATTER_PORT = "8888";

my $dataHash = {
        'Host' => {
            'time' => time,
            'status' => 'OK',
            'ip' => $ip,
            'name' => $host,
            'source' => 'Munin',
            'service' =>\@serviceArray
                }
    };




my $metricname = "";
my @nodes = sendToMunin("nodes");
foreach my $node (@nodes) {
        # get the metics
        debug("node is :: ". $node);
        $dataHash->{Host}{name} = $node;
        my @data = sendToMunin("list $node");
        my @metrics = split(/\s+/, $data[0]);
        verbose(" getting the matrices");
        foreach my $metric (@metrics) {
            my @configLines = sendToMunin("config $metric");
            my $hasCategory = 0;
            my $base = 0;
            foreach my $configLine (@configLines) {
                    print "config line for metric $metric :: $configLine\n";
                    if ($configLine =~ m/graph_category (.+)/) {
                        # $metricname = "$node-$1";
                        $metricname = "$1";
                        $hasCategory = 1;
                            }
                    if ($configLine =~ m/graph_args.+--base (\d+)/) {
                        $base = $1
                              }
                }
            if (!$hasCategory) { $metricname = "others"; }

            print "metric name :: $metricname \n";
            print "base :: $base \n";

            my @metricLines =  sendToMunin("fetch $metric");
            foreach my $line (@metricLines) {

                    my $message =  "OK";
```

```
                    my $field = "";
                    my $value = "";
                    my $now = time;


                    if ($line =~ m/^(.+)\.value\s+(.+)$/) {
                        $field = $1;
                        $value = $2;
                        $message = "OK $metricname $field current value $value";
                        my %service = (
                                'Max', '',
                                'Unit', 'float',
                                'Warn', '',
                                'MeasuredTime', $now,
                                'State', 'OK',
                                'Message', $message,
                                'Min' , '',
                                'Label', '',
                                'Critical', '',
                                'MeasuredValue' , $value,
                                'Type', $metricname.".".$field
                            );
                               push @serviceArray,\%service;


                            }

                }

                }

        my $xmlmessage = createServiceXml();
        debug("XML Format created :: ");
        debug($xmlmessage);

        sendToFormatter($xmlmessage);



    }


exit(0);


sub initializeMunin{

        $muninSocket = new IO::Socket::INET ( PeerAddr => 'localhost',
                                              PeerPort => '4949',
                                              Proto => 'tcp',);
        die "Could not create socket: $!\n" unless $muninSocket;
        $muninSocket->autoflush(1);
        # sendToMunin("nodes");
        my $welcome = <$muninSocket>;
        verbose("welcome :: $welcome");
```

```
        }

# send commands to munin
sub sendToMunin {
            initializeMunin();
            my $cmd = $_[0]."\n";
            my @response;
            verbose("sending command to munin :: $cmd");
            print $muninSocket $cmd;
            # die "socket is dead" unless $muninSocket->connected();


            my $finished = 0;
            while(my $line = <$muninSocket>) {

                        chomp($line);
                        verbose("getting data :: $line");
                        if ($line eq "." ) { break; }
                        else {
                                    push(@response, $line);
                                    verbose("read :: $line");
                                    # if ($cmd =~ m/^list.*/) { $finished = 1 };
                                }
                    }

            close($muninSocket);
            return @response;

        }



sub sendToFormatter(){

     debug("sending message to formatter");
     my $message = $_[0];

     my $gSocket = new IO::Socket::INET (  PeerAddr => $FORMATTER_IP,
              PeerPort => $FORMATTER_PORT,
                      Proto => 'tcp',);
     die "Could not create formatter socket : $!\n" unless $gSocket;
     $gSocket->autoflush(1);
     debug($message);
     print $gSocket $message;
     close($gSocket);


    }


# returns xml from the hash
sub createServiceXml() {

        return $xml->XMLout($dataHash,RootName=>'',NoAttr=>1,KeyAttr => []);

    }
```

```
# Perl trim function to remove whitespace from the start and end of the string
sub trim($)
  {
          my $string = shift;
          $string =~ s/^\s+//;
          $string =~ s/\s+$//;
          return $string;
      }

sub usage {
        print "Usage: \n";
        print "-h Usage \n";
        print "-v Verbose \n";
        print "-d Debug \n";
        print "-1 input file one \n";
        print "-2 input file two \n";
        print "-c column number \n";
        print "./script [-d] [-v] [-h] -1 <file1> -2<file2> -c<columnNumber> \n";
    }

sub verbose {
        print $_[0]."\n" if $VERBOSE;
    }

sub debug {
        print $_[0]."\n" if $DEBUG;
    }
```

A.3 Data Formatter

Listing A.4: Message formatter script for groundworks and graphite

```
#!/usr/bin/perl

# Needed packages
use Getopt::Std;
use strict "vars"; # declare variable
use Data::Dumper;
use IO::Socket;
use XML::Simple;
use Net::NSCA::Client;


# Global variable
my $VERBOSE = 0 ;
my $DEBUG = 0;

# Handle flags and arguments

# Example :: c == "-c", c: == "-c argument"
my $optString = 'vdh1:2:c:';
```

```
getopts($optString,\my %opt) or usage() and exit 1;

# Print  help message in -h is invoked


if ( $opt{'h'} ) {
    usage();
    exit 0;
}




$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};



verbose("Verbose is enabled");
debug("Debug is enabled");



# Main


# Read the xml file

my $xml = new XML::Simple;
my $PORT = 8888;
my $HOST = '10.0.0.6';

my $NAGIOS_LISTNER = "10.0.0.3";
my $GRAPHITE_LISTENER = "10.0.0.2";
my $GRAPHITE_PORT = "2003";


my $LISTNER_SOCKET;
create_listener();
my $client_sock;
while ( $client_sock = $LISTNER_SOCKET->accept()){

    my $clientMessage = "";
    while(<$client_sock>) {
        $clientMessage .=  $_;
    }

    close $client_sock;

    processClientMessage($clientMessage);



    print "\nAt your service. Waiting...\n";
}
# close($LISTNER_SOCKET);
```

```
# process the client XML and format it in the hash
sub processClientMessage() {

    my $clientMessage = $_[0];
    # print $clientMessage;
    my $data = $xml->XMLin($clientMessage,ForceArray => qr{service}x);
    relayData($data);




}


sub relayData {

   my  $data = $_[0];
   sendToGraphite($data);
   sendToNagios($data);
}



sub sendToNagios {

    debug("sending to nagios");

     my $nsca = Net::NSCA::Client->new(
                   remote_host => $NAGIOS_LISTNER,
                   encryption_type => 'xor',
                 );




    my $data = $_[0];

    my $hostname =  $data->{name};
    my $serverBase = $data->{source}."-".$hostname;
    my $message = "";
    foreach my $e (@{$data->{service}})
    {
        my $metric = $data->{source}."-".$e->{Type};
        my $min = $e->{Min};
        my $max = $e->{Max};
        my $t =  $e->{MeasuredTime};
        my $val = $e->{MeasuredValue};
        my $serviceState = $e->{State};
        my $serviceMessage = $e->{Message};
        debug("$hostname $metric $val $serviceState $serviceMessage");

        $nsca->send_report(
            hostname => $hostname,
            service  => $metric,
            message  => $serviceMessage,
```

```
            status    => $Net::NSCA::Client::STATUS_OK,
        );


    }

    return $message;



}


sub formatDataForGraphite {

    my $data = $_[0];

    my $hostname =  $data->{name};
    my $serverBase = $data->{source}."-".$hostname;
    my $message = "";
    foreach my $e (@{$data->{service}})
    {
        my $metric = $e->{Type};
        my $min = $e->{Min};
        my $max = $e->{Max};
        my $t =  $e->{MeasuredTime};
        my $val = $e->{MeasuredValue};
        $message .= "\n$serverBase.$metric $val $t\n";



    }

    return $message;




}


sub sendToGraphite {


    ## format for graphite
    my $data = $_[0];
    my $message = formatDataForGraphite($data);



     my $gSocket = new IO::Socket::INET (   PeerAddr =>$GRAPHITE_LISTENER,
                                            PeerPort =>$GRAPHITE_PORT,
                                            Proto => 'tcp',);
     die "Could not create carbon: $!\n" unless $gSocket;
     $gSocket->autoflush(1);
     debug("sending to graphite ....");
     debug($message);
```

```
        print $gSocket $message;
        close($gSocket);



}




sub create_listener {


    $LISTNER_SOCKET = new IO::Socket::INET (
        LocalHost => $HOST,
        LocalPort => $PORT,
        Proto => 'tcp',
        Listen => 1,
        Reuse => 1,
     );
     die "Could not create socket: $!\n" unless $LISTNER_SOCKET;

     print "At your service. Waiting...\n";


}




# Perl trim function to remove whitespace from the start and end of the string
sub trim($)
{
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

sub usage {
    print "Usage: \n";
    print "-h Usage \n";
    print "-v Verbose \n";
    print "-d Debug \n";
    print "-1 input file one \n";
    print "-2 input file two \n";
    print "-c column number \n";
    print "./script [-d] [-v] [-h] -1 <file1> -2<file2> -c<columnNumber> \n";
}

sub verbose {
    print $_[0]."\n" if $VERBOSE;
}
```

```
sub debug {
    print $_[0]."\n" if $DEBUG;
}
```

Bibliography

[1] collectd The system statistics collection daemon. http://collectd.org/.

[2] open platform for network, application, and cloud monitoring. http://www.gwos.com/.

[3] Graphite - Scalable Realtime Graphing. http://graphite.wikidot.com/.

[4] Jingyue Li, Reidar Conradi, Christian Bunse, Marco Torchiano, Odd Petter N. Slyngstad, and Maurizio Morisio. Development with Off-the-Shelf Components: 10 Facts. *IEEE Software*, 26(2):80–87, March 2009.

[5] I Vlahavas, I Stamelos, I Refanidis, and A Tsoukiàs. ESSE : An Expert System for Software Evaluation. *Knowledge Creation Diffusion Utilization*.

[6] G. Polancic, R.V. Horvat, and T. Rozman. Comparative assessment of open source software using easy accessible data.

[7] GNU. Method for Quali cation and Selection of Open Source software (QSOS) GNU Free Documentation Licence . *Source*, (April), 2006.

[8] Won Jun Sung, Ji Hyeok Kim, and Sung Yul Rhew. A Quality Model for Open Source Software Selection. In *Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT 2007)*, pages 515–519. IEEE, 2007.

[9] Mi-lung Choi, Hyoun-mi Choi, and James W Hong. XM L- Ba sed CO n f i g u ration M a nag em en t for IP Network Devices. *IEEE Communications Magazine*, (July):84–91, 2004.

[10] S. Waldbusser. Remote Network Monitoring Management Information Base Version 2. RFC 4502 (Draft Standard), May 2006.

[11] David Breitgand. On Generic Scalibility Problems in Monitoring Of Data Communication Networks.

[12] Common Information Model. http://dmtf.org/standards/cim.

[13] Web-Based Enterprise Management. http://dmtf.org/standards/wbem.

[14] Remote Monitoring (RMON). http://tools.ietf.org/html/rfc3577.

[15] Andy Bierman, Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. Network Configuration Protocol (NETCONF).

[16] David Josephsen. *Building A monitoring infrastructure with NAGIOS*.

[17] Configuration Management Software For Agile System Administrators. http://cfengine.com/.

[18] Jeff Heflin and James Hendler. Semantic Interoperability on the Web. pages 1–15, 2000.

[19] Frank Van Harmelen and Vrije Universiteit Amsterdam. The Semantic Web : What , Why , How , and When WHY DO WE NEED THE SEMANTIC WEB ? HOW WILL WE ACHIEVE THE SEMANTIC. 5(3):1–4, 2004.

[20] Nagios Is The Industry Standard In IT Infrastructure Monitoring. http://www.nagios.org/.

[21] Monitoring clusters and Grids since the year 2000. http://ganglia.sourceforge.net/.

[22] Liang Zhengyu, Sun Yundong, and Wang Cho-Li. ClusterProbe: an open, flexible and scalable cluster monitoring tool. In *ICWC 99. IEEE Computer Society International Workshop on Cluster Computing*, pages 261–268. IEEE Comput. Soc.

[23] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.

[24] Brian K. Sheffler. The Design and Theory of Data Visualization Tools. *Information Security*.

[25] Khai N Truong, Gregory D Abowd, and Jason A Brotherton. Who , What , When , Where , How : Design Issues of Capture & Access Applications. *Design*, pages 1–20.

[26] Stephen Few. DATA VISUALIZATION PAST , PRESENT , AND FUTURE. *Computer*, 2007.

[27] Melanie Tory and Torsten Möller. Human factors in visualization research. *IEEE transactions on visualization and computer graphics*, 10(1):72–84, 2004.

[28] Marc Green and D Ph. Toward a Perceptual Science of Multidimensional Data Visualization : Bertin and Beyond. *Image (Rochester, N.Y.)*.

[29] Ronald Van Der Pol and Freek Dijkstra. Data Exchange between Network Monitoring Tools.

[30] Michal Džmurá. Introduction to Data Integration Driven by a Common Data Model.

[31] Lucas Zamboulis. XML Data Transformation and Integration A Schema Transformation Approach. (November), 2009.

[32] Maurizio Lenzerini, La Sapienza, Via Salaria, and I Roma. Data Integration : A Theoretical Perspective.

[33] Munin monitoring tool. http://munin-monitoring.org/.

[34] Jan Bergstra and Mark Burgess. A static theory of promises. pages 1–14, 2008.

[35] Www.cfengine.com. Monitoring with CFEngine.

[36] Mark Burgess. Two dimenssional time series for anomaly detection and regulation in adaptive systems.

[37] Nagoya Cabinet: super hyper ultra database manager. http://fallabs.com/tokyocabinet/.

[38] QDBM: Quick DataBase Manager. http://sourceforge.net/projects/qdbm/.

[39] Representational State Transfer (REST). http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[40] RRDtool is the OpenSource industry standard, high performance data logging and graphing system for time series data. . http://oss.oetiker.ch/rrdtool/.

[41] PostgreSQL is a powerful, open source object-relational database system. http://www.postgresql.org/about/.

[42] MySQL is the world's most popular open source database. http://www.mysql.com/products/.

[43] Groundwork Foundation Developer Reference. http://gwfoundation.sourceforge.net/Foundation-bookshelf-2.0.1.pdf.

[44] Farshad Hakimpour and Andreas Geppert. Resolving Semantic Heterogeneity in Schema Integration : an Ontology Based Approach. 2001.

[45] Alon Y Halevy. Why Your Data Won t Mix : Semantic Heterogeneity.

[46] Farshad Hakimpour and Andreas Geppert. Ontologies : an Approach to Resolve Semantic Heterogeneity in Databases. pages 1–25, 2001.

[47] Thomas R Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. pages 907–928, 1993.

[48] Rohit Mahajan and Ben Shneiderman. Visual and Textual Consistency Checking Tools for Graphical User Interfaces. 23(11):722–735, 1997.

[49] Jeremy Mendel and Richard Pak. The effect of Interface Consistency and Cognitive Load on user performance in an information search task. *Human Factors and Ergonomics Society Annual Meeting Proceedings*, 53(22):1684–1688, October 2009.