LOG ANALYSIS TECHNIQUE: PICVIZ

by

SHRADDHA PRADIP SEN

DR. YANG XIAO, COMMITTEE CHAIR
DR. XIAOYAN HONG
DR. SUSAN VRBSKY
DR. SHUHUI LI

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2011

ABSTRACT

Log data that is generated during a number of processes such as Networking, Web surfing, Failures, etc. is quite large. Such log data are supposed to be processed and analyzed so that it can be used to improve the quality of the software, improve its performance, proactive fault detection and handling. There are a number of log analysis techniques that have been presented over the years. Picviz is one such technique.

Picviz is a parallel co-ordinate plot which is used to display huge amounts of data for security purposes. The data can be displayed in multiple dimensions using the parallel co-ordinate system. The primary goal of this software is to ease the analysis of data and finding correlation among the various variables. Since the software deals with huge amounts of data, representing the information all at once creates an image with congested or clustered lines. This makes it difficult to distinguish between lines and obtain any kind of information from the image, which is the main objective of the software. The image that is generated is not clear enough to find or detect any kind of correlation among the various variables. This research work describes two methods (plugins) to solve this problem; the idea is to group lines into sets and represent each set as a single line. This reduces the total number of lines in the figure and makes it easily readable and understandable. The two methods described below are: Grouping based on Comparison of data and Grouping Consecutive data.

DEDICATION

I dedicate this thesis to my parents, Dr. Bina and Pradip Sen for their unconditional love and support throughout my entire life. It is also dedicated to my brother who helped me and guided me in creating this manuscript. Their constant encouragement and support has helped me in fulfilling all my dreams.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank all those people who have helped me in writing this thesis manuscript. I am most indebted to Dr. Yang Xiao, the chairman of this thesis committee, for sharing his research expertise and guiding me through this research project. I would also like to thank all of my committee members, Dr. Xiaoyan Hong, Dr. Susan Vrbsky and Dr. Shuhui Li for their invaluable input and support of both the thesis and my academic progress.

A special thanks goes out to my family and friends for supporting me and also bearing with me while I completed this thesis. This thesis has become possible only because of all your love, support and encouragement.

Thank you all!

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Data is generated from a number of sources such as networking circuits, web surfing, information about the server and the client, access information, hardware devices, failure information, databases, etc. Log files consist of data in a proper format that can be used for analysis. Analyzing such data can be useful in a number of ways such as improving the performance of the system, preventing failures, increase utilization and throughput, increasing reliability and so on. Analysis of the log data is an important step in understanding the basics of the system.

Usually applications now-a-days either search for patterns using a behavioral approach or signature databases [4,5] when considering log files for security but in both the cases information can be skipped. The problem increases with the increase in the size of the log messages. There are various log analysis techniques that have been already presented such as SALSA [8] (Analyzing Logs as State Machine) which is used to examine the system logs from the Hadoop [9] cluster to derive state machine views of the system's execution or the SVM [11] (Support Vector Machine) which uses the system logs to predict event failures.

There are also various log analysis tools that have been designed to analyze other kinds of log data such as CLUEBOX [12] that analyzes performance logs to determine performance problems and issues. Artemis [5] is another such tool which analyzes distributed logs for troubleshooting and performance related issues. It is seen that each technique or tool deals with only a particular type of log data and is unable to analyze other types of log data. Another drawback of these techniques and tools is that data is not represented visually. Such a representation makes the process of analysis easier for a human analyst.

Picviz [2] was mainly designed to analyze network log data collected from various network sniffers or network analyzers and represents the log data in a visual format. This software can also be used to analyze various other types of log data also such as system logs, error logs, performance logs and so on. It is not limited to a particular type of log data. The Picviz software has been described in detail in the later section. There are still some problems or challenges related with the Picviz software [2] which are described in detail in this paper. The most basic challenge is the presence of dense or clustered lines which renders the image unreadable. This makes it difficult to view the entire data for a line. Since most of the lines overlap each other so does their text labels and it becomes unreadable. The software provides the option of filtering data but this will generate an image that consists only of particular set of data line such as for IPaddress = 190.160.35.1 or for Protocol != TCP. This option will not generate the entire information present in the log files. This work tries to provide a solution for this problem.

One way to solve this problem is to reduce the number of data lines that are generated. This can be done by grouping together data lines to form a set. Each such set is then represented by a single line in the parallel co-ordinate plot. This will essentially reduce the number of lines generated in the plot and also avoid overlapping of lines. Grouping can be done based on the occurrence of same data line over and over again in the log data with different time stamps. Another way is to group the lines over a particular time period say from 10:01:56 to 10:02:16. These methods will still represent all the data that is content in the log files but reduce the number of data lines that are generated in the figure.

2. LOG SOURCES

2.1. Disk Drive Workload Logs

The data generated from the disk drives is used as input log for the workload characterization in [4]. This data provides general insight on the disk drive workloads over the Cheetah 10K family and the Cheetah 15K family [4]. The disk drives contain attributes which describe the various operations that are undertaken by the drive. These attributes include the total number of bytes read and written in the drive, the total seeks performed, the total number of drive spin-ups and the total number of operation hours. The set of attributes is updated in the disk drive during the system idle times. These attributes are obtained from the disk drives which are returned back to the producer due to some kind of fault. The returned disks undergo Field Return Incoming Test [4] this provides the producer with a set of data. This data can then be used to find the workload that the disk drive was exposed to during its run in the field and also to construct average behavior and the worst case scenario. The workload characteristics can be used to design feature which tends to improve the overall drive operation, performance, power consumption and reliability.

In this case the attributes describing the amount of bytes read, written and seeks completed is used for analysis. During analysis this data set is further divide into two subsets according to their age: the drives that have been used for less than a month and the drives that have been used for more than a month. Empirical cumulative [4] distribution of the age is constructed for the data sets of the drives. There are a few drawbacks in this data set: firstly it does not help in understanding the dynamics of the workload demands over time and secondly it cannot be confirmed that the data obtained is from a random set of drives. For the workload

characterization the attributes are analyzed and conclusions are drawn from it. The spin-ups attribute is studied to find out the number of times the drive has been powered up/down. This information is useful since it is expected that drives are supposed to be operational all the time and are mostly never shut down, except during it early life time when it is getting adjusted to the field usage. The read and write which are the most important attributes are used to determine the amount of data requested from the drive and the average amount of work processed by the drive. It is seen that the amount of data written into the drives is more during the first month of operation as compared to the later months. The empirical distribution of this attribute allows understanding of the worst case scenario regarding the utilization of the drives. It is also concluded that the larger capacity drives are installed in the systems where more data access per unit time is required as compared to the systems containing the low capacity drives. The read/write ratio [4] is used to determine whether the disk drive is read oriented or write oriented and depending on it the performance and the reliability of the system is determined. The attribute completed seek time describes the number of requests that are completed by the drives and is more closely related with the utilization of the disk drive. Since, the head positioning overhead that often dominates disk service times is associated with the number of request.

This data provides a good source of information when the disk drive operations in the field are considered. As mentioned earlier the high-end drives Cheetah 10K and Cheetah 15K are used. The data consists of approximately 110,000 Cheetah 15K drives and 200,000 Cheetah 10K drives [4]. Data set consisted of cumulative amount of bytes READ and WRITTEN by a single drive and also the number of hours it had been in the field, the number of spin ups the drives performed as well as the total number of completed seeks. This information was used to extract the average performance of a drive family which is generally linked to a specific computing

environment. To understand the worst case scenarios for the disk drives with respect to the load seen by the families, the distribution of the obtained attributes was constructed. The differences and commonalities of these sub-families were evaluated. These characteristics can be used to develop new techniques and features in the disk drive that aims at increasing the drive performance, power consumption and reliability.

Drive Family	Total Drives	Less than one month old	More than one month old
Cheetah 15K	108,649	19,557	89,092
Cheetah 10K	197,013	43,999	153,014

Table 2.1: Size of data set [4]

Efforts to design advanced storage systems are dependent on the knowledge gained from the workload characterization and the data set represents a good source of information with regard to the disk drive operations in the field.

2.2. Telemetry Data

A lot of data is generated which needs to be diagnosed and analyzed by the researchers. Since the distributed systems has become extremely dependent on the system, network and each other, due to this there arises the problem of accessing the data required by the researchers. Even if access

to such data is gained it becomes difficult to co-relate all the data together since each is in a different format and the vastness of the data also hinders the ability of the researcher to use it easily. It is difficult to diagnose and analyze the data that is present in the distributed systems mainly because of its distributed nature and also due to the incredible amount of data that is stored in the systems. CyDAT (Cyber-center for Diagnostics Analytics and Telemetry) [10] is established within CyLAB at Carnegie Mellon to address this problem of data diagnosis. Data logging is not only done in the virtual systems but also in the physical systems and environments. A lot of sensors have emerged during the recent time which generates a lot of data. This increase in sensing and telemetry capabilities has been converted into the problem of data maintenance.

Data is generated from embedded sensor systems, operational systems, experimental sensors and laboratory environments. All this data needs to be collected systematically and managed so that it can be easily used by the researchers to utilize it for their experimentation. CyDAT is working towards creating an advanced, open and collaborative instrument to support the research into management, collection and also use of advanced telemetry and analytics in a multi and inter-disciplinary fashion. The system can accommodate and integrate telemetry data, analytic algorithms and response mechanisms for a number of disciplines to cause improvement in the field of data collection and analysis. CyDAT [10] aims at decreasing the data collection and management burden of the researchers and provide a way to expand the realm and reach of data possibilities. The data for this is collected from a number of IT enabled observation points and making it available to the researchers. The main difficulties into building such a system are: access, multiple formats, scale, proof of the diagnosis and sharing knowledge. To solve the access problem the data was collected at real time and sent to a common storage facility. Specific

anonymization methods and file/directory permissions where used to preserve privacy while providing valuable information at the same time. For solving the problem of different data formats they built a common event record (CER) [10] instead of waiting for the standards to consolidate. Scale was one of the primary challenges in building the system. For the system the group set the proof-of-concept value to 5K events/sec for initial experimentation. A new taxonomy was used to automate the way of exporting the captured knowledge of the expert diagnosticians and send it to the other members of the team. They constructed systems with the following capabilities [10]:

- The system can process flow records at a high rate of over 8000 events/sec.
- It can transform groups of flow events into a consolidated record that enables better correlation.
- Remove the attributes that are of little value to the end analysis tool.
- Systems that build higher order events that identify the n hosts that use the most network bandwidth, packets and flows.

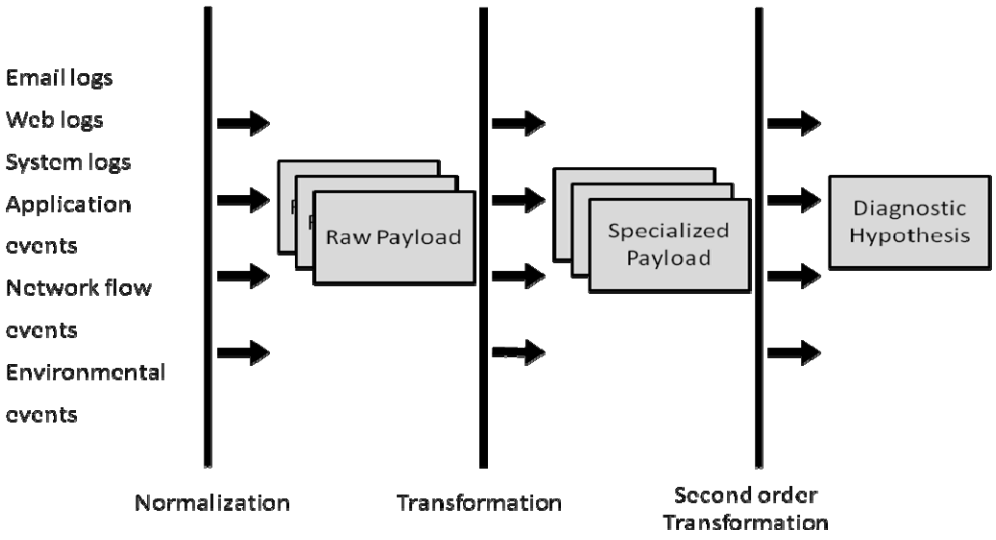

Figure 2.1: Representational overview of event collection, transformation/reduction and analysis

The intermediate events could be used as input for the security applications and also for a myriad of purposes by the researchers (see Figure 2.1 [10]).

Since real time log values which provide significant additional flexibility to the current routine were used for the model so it was designed from a data-flow perspective. The model uses pipeline pattern where data sources are immediately normalized into a common event form and then may be transported across a backplane of agents to optimize the flow of information to target analytics. The core elements of the architecture: normalize, select, transform, project, route, store, query, analyze and application proxy are considered as the basic capability for the event backplane.

EDDY (End-to-end Diagnostic Discovery) [10] was created as a reference implementation of the experimental core around which the testing could be done along with the integration of the new concepts brought in by the use cases. EDDY is used for management, exchange and correlation of the event and log information. It defines a method for efficient transfer of the local event information from the sensors to data managers to analyzers. The EDDY architecture was designed to be simple efficient and extensible. The EDDY architecture is based on the following four assumptions:

- A system should be flexible enough to be used by a small installation as well as a large distributed network.
- The system could be easily updated without any catastrophic changes taking place in it.
- The existing analytic techniques can leverage the data orchestration function that is provided but the main aim is to enable those techniques to be composed, modified and extended to consider a variety of information.

- A working code is to be developed which address all the above mentioned implementation specifics.

Following are the basic components and format of the EDDY CER architecture that is used [10]:

- Minimal common event elements
  - CER version
  - Reference identifier for every record
  - Timestamp for creating event
  - Location where the event was seen
  - Location where the event was introduced into the backplane
  - Different types of events
- Additional common elements
  - Gravity of the event
  - User tag
- Encapsulation of the transported events
  - Opaque encapsulation of external schema
  - Ad hoc schema
- Event representation
  - The event elements are defined by simple data types
  - Common diagnostic objects

Following are the elements of the EDDY [10] transport architecture:

- The Event Channel
- The Query Channel

Security is an important consideration in this case since complete knowledge of the working of the system can be dangerous in the wrong hands. Security ramifications are needed while collecting the data and correlating them. New methods are required to reduce the risk factor in such a system and methods are required to provide proper security for these systems. One way to do this is, to anonymize the data so that trends can be identified without exposing the individual components to the targeted attack which would minimize the risk.

Presently analysis of DNS lookups, syslog records, Shibboleth logs, web access, environmental sensors, embedded system events and many more are being performed using this system. CyDAT [10] is a real world system that is being used to diagnose and analyze the real-time data for active and experimental systems across many areas.

3. LOG ANALYSIS TOOLS AND TECHNIQUES

3.1. Error Logs

Error logs are the log files containing error information which is used for analysis. Such log data can be used for failure prediction of the system. In this case the errors that have occurred in the past are used to determine whether similar kind of errors will occur in the future or when it is likely that an error will occur in the system. In the paper the author uses the data set obtained from a commercial telecommunication system to predict failures occurring in the future. The telecommunication system log files are not machine readable so it needs to be converted into machine readable format before it can be processed. The figure 3.1 [9] shows the shows three lines of log record generated by the telecommunication system.

Following transformations were applied on the data to obtain machine readable log files [9]:

- Eliminating log file rotation
- Identifying borders between messages
- Converting time

These error messages were also provided with unique IDs for the type of error occurring in the system. It also helps in reducing the amount of data that needs to be analyzed and thus making the task more efficient. The data is used to train the HSMM [9] based failure predictor and for this, two types of data sequences are needed: a set containing failure related error sequences and a set of non-failure related sequences. These data sequences are extracted from the log files involving three parameters: lead time, data window size and margins for non-failure sequences. For the experimentation purposes they used the lead time value as five minutes, also

the data window size had a length of five minutes. The margin for failure was found out to be twenty minutes. The sequences containing similar types of errors are grouped together using failure sequence clustering. For doing this firstly dissimilarity matrix is obtained for the given data set and the models which are to be trained using this data set. Then a hierarchical clustering algorithm is applied on this data set to obtain groups of similar failure sequences. Analysis of the sequence clustering is done to find the influence of the various parameters in the data on sequence clustering. The data thus obtained cannot be directly provided to the HSMM for failure prediction since it contains noise or useless data generated during the transmission in the telecommunication system. The noise that is generated is removed using a filtering mechanism. The data that is finally obtained is provided to the HSMM for failure prediction. The data that is generated is initially used to train the HSMM so that whenever it is given a similar set of data it can determine the probability that an error might occur in the future. It is essential to perform such elaborated data preparations on the log files to achieve good prediction accuracy. Such data preparation improves the accuracy of the system and increases its efficiency. The following techniques were applied on the data [9]:

- An algorithm to automatically assign IDs to the error messages
- A clustering algorithm to group similar error sequences
- A statistical filtering algorithm to decrease noise in the sequence.

In this case telecommunication data is being used for failure prediction but there are a number of different types of data sets that can be used for failure prediction but it is very necessary that the data should be in proper format so as to get accurate results from the set of data.

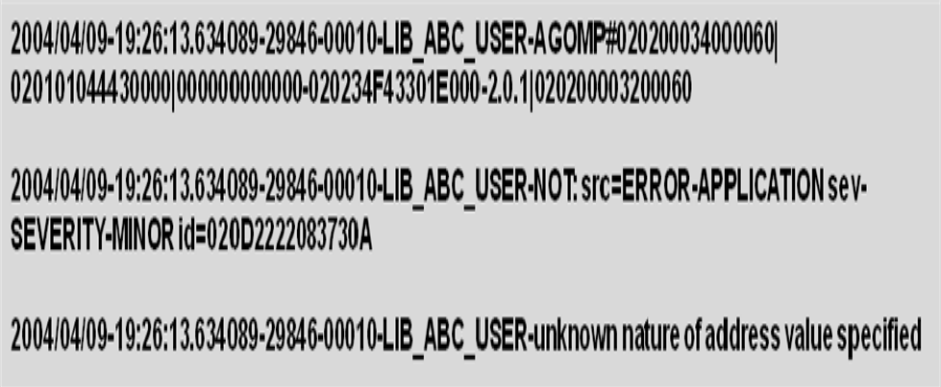

Figure 3.1: Three lines of log record generated by the telecommunication system [9]

3.2. System Logs

System log files have been used to predict system failures in a number of cases. Below is described another such method in which the syslog has been used in a new spectrum-kernel Support Vector Machine (SVM) [11] to predict event failures. This method uses a sequence of or pattern of messages to predict failure. A sliding window of messages is used to predict the likelihood of failure. The frequency representation of the message sub-sequences observed in the system logs are used as input for the SVM. SVM is used to associate the input to a class of failed or non-failed systems. System logs from a Linux based computer cluster was used as the input for the proposed spectrum kernel SVM.

It is hard and requires a lot of time and resources for the recovery process of a High Performance Computing (HPC) [11] system in which failure was caused due to hard disk failure or processor failure. The reliability of the system reduces with the increase in the size of the system since larger systems have more chances of failure. Predicting system failures and scheduling applications and services around it can help increase reliability of the system. This method can be used to predict critical system events based on the Support Vector Machines (SVM) [11]. A training set of data called the support vector needs to be provided for training the

SVM system. The average number of message during a period of time or in other words the aggregate features are used for SVM based classification. Also, the sequential nature of the messages is an important feature of the system logs. Spectrum representation of the message sequence using an n-length sequence of messages is used as one of the features for SVM classifier [11]. The SVM uses different message sequences to classify the systems as either fail or non-fail systems. SVM is used for predicting by concluding that a series of messages are or are not associated with the failures occurring in the future.

System logs are one of the most important files that are required for managing computer systems since they provide information such as history or audit trail of events. In this case, an event is the change in system status, such as application failure, user login, system failure or security problems that have occurred in the system. Such type of forensic analysis is very useful but the data contained in the system log files can also be used for failure prediction. System logs are created by different processes being executed on the system such as general information about user login or more critical warnings about system failures. These data are provided to various standard machine learning techniques such as Hidden Markov Models (HMM), Partially Observed Markov Decision Problem (POMDP) and Bayes network [11] for predicting failure in the system. Specific sequences of these logs provide enough information for the prediction of failure but due to the large amount of data that is generated or stored in the system log files it becomes very difficult to find such a specific pattern or sequence in the data. The applications send the message in format of text files to the logging services where they are stored in the system log files in the same format. The messages sent by the application files are stored in the text files in the order which they arrive at the syslog process. In case of clustered network the syslog process resides on a separate host and the messages are sent to it from each node in the

network over a secure network connection. The main responsibility of the syslog process is to manage the log files while the message content is created by the application process. Figure 3.2 [11] represents the specific format of the syslog files consisting of six fields: host, facility, level, tag, time, and message.

The host field is used to describe the IP address of the system that generated the message. The facility field describes the source of the message that is from kernel or user space. The level field is used to indicate the intensity of the message that is whether it is general or critical information. The tag field is always a positive integer that describes the facility and level fields such that lower tag values represent more critical information. For example as shown in figure 3.2 [11] the tag value of 1 represents a disk failure whereas a tag value of 189 represents execution of a command. Since the tag value is a combination of facility and level fields, it does not represent the actual message content. For example, it is seen in figure 3.2 [11] that the tag value 38 occurs twice but the message contents are different.

Host	Facility	Level	Tag	Time	Message
192.129.8.6	local7	notice	189	1171061732	Systat
192.129.8.6	kern	info	6	1171061732	Kernel md: using maximum available idle IO bandwidth
192.129.8.6	cron	info	78	1171061733	Crond 2500 (root) CMD (/usr/lib/sa/sa1 1 1)
192.129.8.6	auth	info	38	1171062445	Rsh(pam_unix) 2215 session opened for user by (uid = 0)
192.129.8.6	auth	info	38	1171062445	In.rshd 2216 root@hpcs2.cs.edu as root: cmd=/root/temps
192.129.8.6	daemon	info	30	1171062590	Smartd 88 Device: /dev/twe0 SMART Prefailure Attribute
192.129.8.6	auth	notice	37	1171062590	Sshd(pam_unix) 12430 auth failure; logname-e1-fork-o
192.129.8.6	kern	info	6	1171062590	Kernel md: using 512k, over a total of 12287932 blocks
192.129.8.6	cron	info	78	1171062601	Crond 2500 _root) CMD (/usr/lib/sa/fork-it 1 1)
192.129.8.6	kern	alert	1	1171062690	Kernel raid5: Disk failure on sdel, disabling device

Figure 3.2: Example syslog messages, each consisting of six fields [11]

The time field represent the time the message was sent to and recorded by the syslog process. The message field describes the event that occurred in the text format. The message field created by the application is a free form text field which makes the analysis more complicated and difficult requiring complex natural language parsers.

Self-Monitoring Analysis and Reporting Technology (SMART) [11] are a type of syslog messages that provide information regarding the disk drive health. ATA and SCSI hard disks are using SMART boards as their standard component. SMART disk drives are used to monitor the internal health of the hard disk drive and warn if there are chances of some problem. Sometimes the hard disk itself provides advanced warning regarding the disk failure long before it actually happens. SMART allows administrators to perform self-tests and monitor the system performance and reliability attributes. The smartd daemon used to regularly monitor the disk's SMART data to check for any signs of problem. The smartd registers the system's disk then checks their status after every 30 minutes to check for attributes that may indicate error. The daemon sends information to the system log file if there are signs of failing health or increased number of errors or failed self cases in the system's disk. The information generated by the SMART parameters is sufficient to find the drive failures but are insufficient for prediction.

The SVM [11] approach was evaluated using actual system log files that were collected over approximately 24 months from 1024 node Linux-based computer cluster. The systems were similarly configured and consisted of multiple disks and processors. The log that was used consisted of over 120 disk failure events that were separated over a period of minimum 24 hours. The log files consisted of an average of 3.24 messages per hour for every system and thus, each machine averaged approximately 78 messages per day. The range of tag values was from 0 (for critical messages) to 189 (for least important message) and consisted of 61 unique tags. 1200

messages over a period of 15 days was collected from the systems that did or did not experience disk drive failure and the systems that did experience disk failures, the last message was the disk failure message. A subset M of the given set of 1200 messages was used to make prediction. The length of M could be any of the following: 400, 600, 800, 1000 or 1100 messages. Larger the value of message input better the failure prediction since more data is available.

Log files generally contain useful information about system failure. The system administrators use the information stored in the file records to determine the cause of critical events. The log file analysis is done after an event has occurred and this is being used increasingly to predict events. System failures result in higher operation cost and lower productivity. Accurate prediction of system failures help in better management of computer resources, applications and services. So it is necessary to predict system failure to improve the system performance.

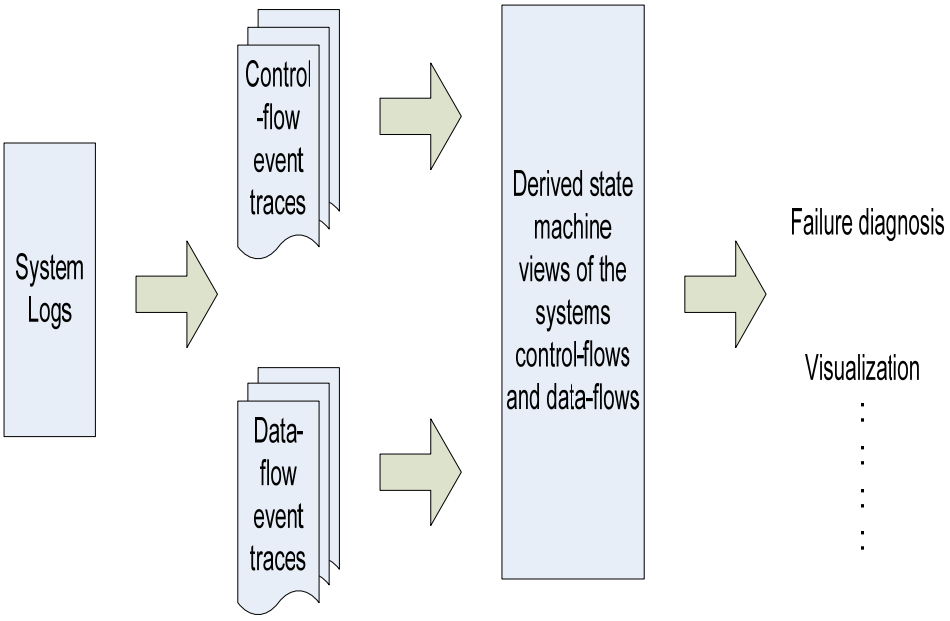

Figure 3.3: Overview of SALSA system [8]

3.3. SALSA

SALSA (Analyzing Logs as State Machine) [8] is used to examine the system logs from the Hadoop cluster to derive state machine views of the system's execution. It is also used to derive the data flow model, control flow and related statistics. Higher level useful analysis can be effectively generated by further exploiting SALSA's derived views. SALSA's value is illustrated by developing failure-diagnosis and visualization techniques for three different Hadoop workloads, based on the derived state machine view.

SALSA was designed to automate the system log analysis. It involves examining logs to trace data flow and control flow execution in a distributed system. It also aims at deriving views similar to the state machine views, of the system's execution on each node. Figure 3.3 [8] display the overview of the SALSA.

The log data accuracy depends on the programmer implementing the logging points in the system, so only the state machines executing within the target system can be inferred. The verification of whether the derived machines faithfully capture the actual log data that executes within the system is not performed. Instead, the derived state machines are leveraged to support different kinds of useful analysis such as [8]:

- Understanding or visualizing the system's execution.
- Discovering the data flows in the system.
- Finding bugs in the system.
- Localizing the performance problems and failures.

SALSA is a log analysis technique that aims at deriving state machine views from unstructured text based logs to support failure diagnosis, visualization and other uses. SALSA's

approach is applied to the log generated by Hadoop [13], the open source implementation of Map/Reduce [14]. The contributions of this paper are:

- A log analysis technique that is used to extract state machine views of distributed system's execution along with its data flow and control flow.
- A usage scenario which is beneficial in preliminary failure diagnosis for Hadoop using SALSA.
- Another usage scenario which enables the visualization of Hadoop's distribution behavior using SALSA.

SALSA [8] does not require any changes to the operating system, middleware or the host application. For example, consider the example given below to understand SALSA's high level operation:

A distributed system with many consumers represented by B and producers represented by A, running on any host at any point in time is given. Now, track the execution of two task A1 and B1 on host Y and A2 on host Z as captured by a sequence of time stamped log entries at host Y:

[t1] Begin Task A1

[t2] Begin Task B1

[t3] Task A1 does some work

[t4] Task B1 waits for data from A1 and A2

[t5] Task A1 produces data

[t6] Task B1 consumes data from A1 on host Y

[t7] Task A1 ends

[t8] Task B1 consumes data from A2 on host Z

[t9] Task B1 ends

:

It is seen from the above log entries that the executions of A1 and B1 are interleaved on Y. It is also seen that the log capture a data flow for B1 with A1 and A2. SALSA derives interesting statistics, such as the duration of states, which is then compared along with the sequence of states across hosts in the system. It is also used to extract data flow models which is useful to visualize and examine any data flow dependencies or bottlenecks that cause failures to increase in the system. The main focus of this paper was on the analysis that can be performed on logs in their existing form.

Hadoop [13] is an open source implementation of Google's Map/Reduce [14] framework that enables parallel, distributed and data intensive application by dividing a large job into smaller tasks and also divides a large data set into smaller partitions. This enables each task to be processed in parallel on a different partition. The main abstractions of this implementation are [13]:

- **Map:** To generate a set of intermediate results, Map tasks that processes the partitions generated from the data set using key/value pairs.
- **Reduce:** It tasks that merge all the intermediate values belonging to the same intermediate key.

HDFS (Hadoop Distributed File System) [13], an implementation of the Google File System is used by Hadoop to share data among the distributed tasks in the system which divides and stores files as fixed size blocks. Hadoop has the master-slave architecture with a single master and multiple slaves. The master host runs two daemons: JobTracker and NameNode while the slave host runs two daemons: TaskTracker and DataNode. JobTracker is used to

manage and schedule all of the tasks belonging to a currently executing job. NameNode is used to manage the HDFS namespace and regulates access to files depending on type of clients. TaskTracker is used to launch tasks on its host based on the instructions of the JobTracker and keeps track of the progress of each task on respective hosts. DataNode is used to serve data blocks to HDFS clients.

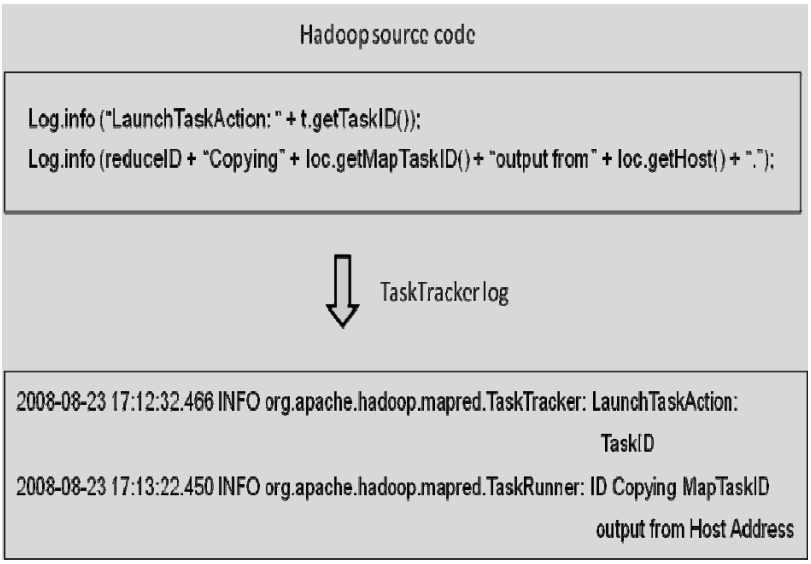

Figure 3.4: log4j generated TaskTracker log entries [8]

The logs of Hadoop's execution are captured by the Java based log4j logging utility for every host. log4j is a commonly used technique to develop log entries. It enables the developers to generate log data by inserting statements in the program code at various points of execution. A separate log for each of the daemons: JobTracker, TaskTracker, NameNode and DataNode by default, is generated by Hadoop's log4j configuration. Each log is stored on the local file system of each executing daemon respectively. Syslog records events, exceptions and error messages in the system. Hadoop's logging framework checkpoints execution since it captures the execution

status of all Hadoop tasks and jobs on each and every host. Time stamped log entries having a specific format are generated by the Hadoop's default log4j configuration. Figure 3.4 and 3.5 [8] show the TaskTracker and DataNode log entries that are generated.

SALSA mainly focuses on the log entries that are generated by the Hadoop's TaskTracker and DataNode daemon. There is increase in the number of the daemons with the increase in size of Hadoop cluster, which makes it very difficult to analyze the set of logs manually that are associated with it. Therefore, for SALSA's automated log analysis TaskTracker and DataNode are selected. Each TaskTracker log at a high level, records information related to the TaskTracker's execution of Reduce and Map on its local host, and also any dependencies between locally executing Reduce and Map outputs from other hosts. DataNode log records information related to the writing and reading of Hadoop Distributed File System data blocks which are located on the local disk.

For each Map or Reduce task on its host, the TaskTracker spawns a new JVM. Every Map thread is inter-related with some Reduce thread where the Map's output is used-up by its related Reduce. When the Map task's output is copied from the host to the host executing the related Reduce, the Reduce and Map tasks are synchronized to ReduceCopy and MapReduceCopy events. SALSA derives the control flow where Maps on one host can be synchronized to Reduce on a different host, across all Hadoop hosts present in the cluster by parsing all of TaskTracker logs belonging to the hosts collectively. Based on the TaskTracker log, SALSA derives a state machine for each unique Map or Reduce in the system. A state corresponds to each log delineated event within a task.

The DataNode daemon mainly runs three different types of data related threads:

- ReadBlock that provides blocks to HDFS clients.

- WriteBlock that obtains blocks which are written by HDFS clients.
- WriteBlock_Replicated that obtains blocks that are written by HDFS clients which are subsequently transferred to another DataNode for replication.

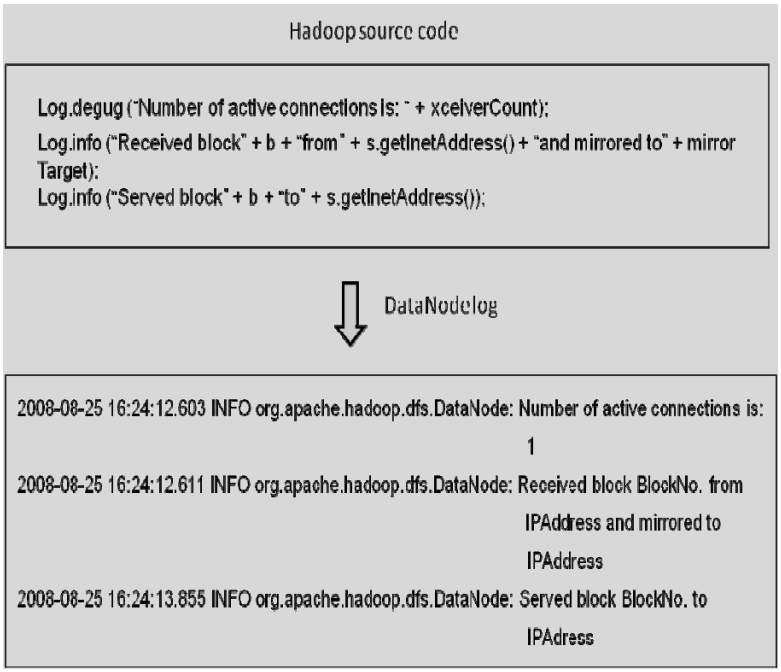

Figure 3.5: log4j generated DataNode log entries [8]

It spawns a new JVM thread for each thread of execution and also runs on its own independent JVM. SALSA derives a state machine based on the DataNode log on each host for each of the unique data related threads. Within a data related thread each log delineated activity corresponds to a state. Depending on SALSA's derived state machine approach multiple statistics from the log data are extracted for the states: Map, Reduce, ReduceCopy and ReduceMergeCopy [8]. They analyzed traces of system log from 5 slaves and 1 master node Haddop 0.12.3 cluster. Every host consisted of an AMD Opeteron 1220 dual core CPU, gigabit Ethernet, 4GB memory, 320GB disk and ran amd64 version Debian/GNU Linux4.0. Three

different workloads were used of which two are commonly used to benchmark Hadoop: Sort, Randwriter, Nutch [8]. Hadoop jobs that are approximately 20 minutes long are present in all the experimentation iterations. The logging level was set to DEBUG. The logs of Hadoop system was cleared before every experiment iteration was performed and collected after every experiment iteration was completed. To show the usage of SALSA in failure detection in Hadoop, a failure was introduced into one of the five slave hosts in between each of the experiment iteration. Two candidate failures were selected from the list of real world Hadoop problems reported by the developers and users to demonstrate the use of SALSA for failure diagnosis.

SALSA [8] analyzes syslogs to derive distributed data flow models, control flow, state machine views and system execution statistics. These different views of log data can be used for different purposes like failure diagnosis or visualization. SALSA is applied to the Hadoop log to diagnose documented failure and to visualize its behavior. The derived data flow models were superimposed on the control flow models for diagnosis of correlated failures.

3.4. Performance Logs

Performance logs are recorded from the software systems and define a large number of counters measuring number of operations per second, latencies, number of seeks performed. The data is similar to the data obtained from the disk drives but in this case it is obtained from software instead of the hardware. The performance logs are periodically dumped into complex storage controller software consisting of counters. The counter may be correlated with each other and evolve over a long period of time. These counters are updated by different components at different times and frequencies. As mentioned in the paper the CLUEBOX [12] uses such data

for determining the performance of a given workload, identify a performance problem and identify the anomalous counters in a system. The workload characteristic and intensity are two dimensions of the workload which are analyzed using this tool. The attributes read/write ratio, sequentiality and arrival distribution describe the characteristic of a work load, whereas intensity refers to attributes such as arrival rate of operation and throughput of data read.

The dimension of the data that is present in the storage is quite large so it should be reduced before any further analysis can be performed on it. Dimension reduction technique is applied to this large data as a data preparation step. The authors of the paper use Principle Feature Analysis (PFA) as the data reduction technique due to which there is nearly 80% reduction in the number of counters. The workload also consists of workload signature which further reduces the number of counters that need to be analyzed and provides a concise description of the workload. The tool uses a machine learning technique called as random forest (RF) [12] which provides the ranking of the counters by their importance. Such ranking is necessary since there are some counters that are more important in describing the workload as compared to others. Clustering is performed on the increasing subsets of the ranked counters to obtain the smallest set of counters that describes the workload more effectively. In this way the counters having low information value is removed from the set of useful counters. This set of counters is called as workload signature profile [12]. The signature of a workload is the medoid of the cluster as calculated above. In case of anomaly detection the same data cannot be used to determine the problem since the value of counters is changed when the anomaly occurs. To solve this problem the value of the counter is required when the system was working properly for comparing it with the present data containing the anomaly. For obtaining the current workload the initial workload is compared with all the known workloads. The medoid of the workload that

has the highest proximity to the test load is considered the closest workload cluster that is used as the current workload. The test load is used to predict the average latency using the decision tree of the random forest. To find the anomalous counters the observed latency is compared with the predicted latency. The sensitivity of latencies to each of the counter values and the degree of deviation between the expected and observed values of the counters are the two dimensions of the anomaly detection. If there is major deviation in the top counters then it is a probability that they are important anomalies. To identify the counter which is most affected, CLUEBOX can be used to examine the counters depending on the errors between the expected value and the observed value. The error and importance values of the counters are taken into account to get the ranked list of anomalous counters which could help in identifying the root cause of the performance problem. The following methods are described below, to determine performance problem in a system: inferring workload signatures from performance logs, matching new workloads against known signatures and narrowing down the root cause of anomalous behavior. A number of different kinds of workloads can also be used in this tool such as temporal workloads, mixed workloads, background workloads or derived workloads.

3.5. Artemis

Artemis [5] is a modular application designed for troubleshooting and analyzing the performance of large clusters using its distributed logs. It is composed of four modules:

- Distributed log collection and data extraction
- Database storing the collected data
- Interactive visualization tool for exploring the data
- A plug-in interface allowing users to implement data analysis tool such as:

- extraction and construction of new features from basic measurements collected
- implementation and invocation of statistical and machine learning algorithms and tools

A case study is presented in the paper [5] using Artemis to analyze a Dryad application running on a 240 machine cluster. Management and analysis of distributed logs is an important part of meeting the performance and dependability goals of clusters. Aggregation of log information from the machines in the cluster is required for understanding performance. To infer system interactions debugging correctness problem requires the correlation of log information. Performing tasks such as visualization and feature extraction are required for automated analysis using statistical machine learning techniques. All the above mentioned elements are incorporated in Artemis. Artemis separates application specific parts from generic application independent parts and also separates data analysis from data collection. The flexibility of Artemis makes it customizable for the domain of distributed Dryad [15] application. Artemis is used for debugging a Dryad application running on 240 machine cluster. Artemis has also been used for exploring datasets produced by other distributed applications such as telemetry performance data for Windows and performance measurements from a data centre providing enterprise services. It is an application that focuses on the analysis of distributed logs and presets the following unique combination of characteristics [5]:

- All the important tasks required for log analysis such as collection, storage, visualization, analysis are integrated into a single tool by Artemis. It was designed to be a single stop for the programmer attempting to understand the performance of a distributed application.
- It is extensible and modular in several ways:
  - Multiple data sources and data types can be manipulated by Artemis.
  - Generic and user defined data analysis are performed by Artemis.

- Keeping the humans in the data analysis loop, Artemis was built around a Graphical User Interface.

A lot of work has been done in the field of distributed log collection, visualization and data analysis. Artemis is a unique tool that integrates all these features into a single tool in a generic and extensible architecture. Artemis can be used to analyze the performance of various distributed computing applications.

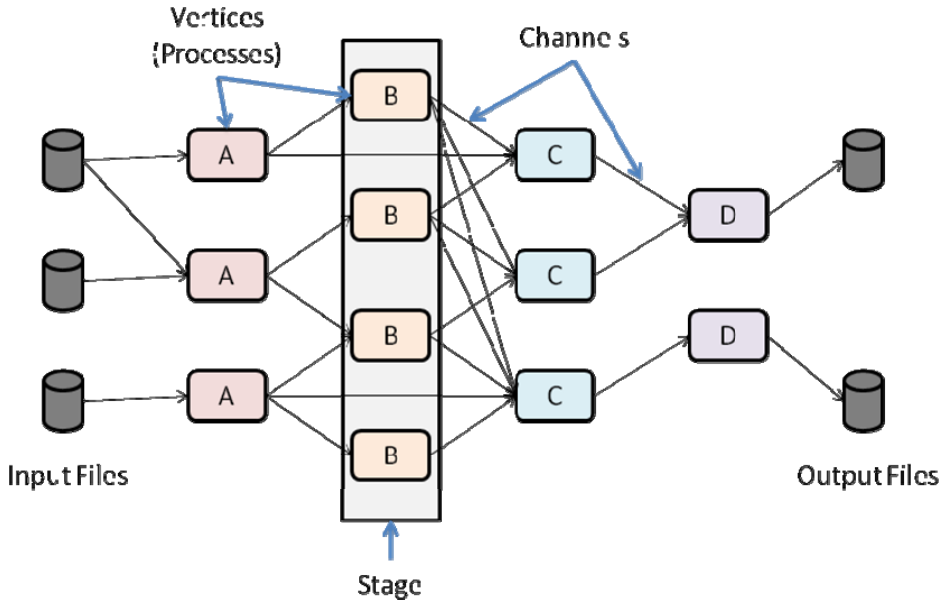

Figure 3.6: Structure of Dryad job

Dryad is a middleware used for building data parallel distributed batch application. Figure 3.6 [15] depicts the structure of Dryad application.

A job consists of set of stages which are composed of arbitrary number of replicas of a vertex. The edges of the graph represent the point to point communication channels. The graph is required to be acyclic. The communication channels consist of arbitrary data items in a finite

sequence. Each vertex of the graph corresponds to a single process, but many vertices connected with shared memory channels can be run as separate threads in the same process. The model of Dryad is simple and powerful. Dryad runtime is unaware of the semantics of the vertices still it can provide a great deal of functionality such as generating job graph, handling transient failures in the cluster, scheduling the processes on the available machines, visualizing the job, collecting performance metrics, invoking user defined policies and dynamically updating the job graph in response to the policy decisions.

The structure of the Artemis distributed log analysis toolkit is shown below in figure 3.7 [5]. Artemis cleanly separates the application independent parts from the application specific parts. By replacing or modifying the application specific parts Artemis can be adapted for analyzing a new distributed application.

There are four main parts of Artemis [5]:

- Log collection, persistence, filtering and summarization
- Data storage
- Data visualization
- Data analysis

The data is aggregated from the following sources for analyzing Dryad jobs:

- Job manager logs
- Logs of the vertices generated by the Dryad runtime library
- Logs from remote execution cluster services
- Performance counters
- Logs from the cluster name server describing the cluster network topology.

The log files can be text files, XML or binary coded. Additional sources of information can be added easily such as SNMP logs from the cluster. A Dryad [15] job running on a large cluster composed of tens of thousands of processes for tens of minute can emit in excess of 1TB of logs. Each Dryad vertex has its own private home directory where it maintains its working space and dumps its logs. Around 1MB/s/process is produced by each vertex. After the completion of a Dryad job, the cluster level scheduler garbage collects from the private home directory of the job vertices after some fixed time interval. An additional application specific GUI front-end for interfacing Artemis with the Dryad cluster scheduler was built. This GUI can be used by the data analyst to go through all the jobs present on the cluster and to select which ones to copy and analyze.

Artemis is used to parse the job manager logs which contain global per job information including pointers to the location of all the vertex logs. The information thus obtained is used to prepare the input for a distributed DryadLINQ [16] computation. DryadLINQ is used to copy, locate, read, parse and summarize all the distributed data sources which reside on the machine in the cluster. Artemis data collection can take advantage of the parallelism available in the very cluster under study for the data intensive parts of the analysis by using DryadLINQ. DryadLINQ computation is used to spawn a vertex on each machine containing interesting logs.

On the 240 machine cluster containing several gigabytes of log data requires only a few minutes for filtering. Dryad vertex uses a machine for a bounded time in contrast to the cluster services and performance counter collection tasks are running and logging continuously on each machine in the cluster. Only the relevant portions of the service logs for every machine are extracted by the Artemis log collector. The extracted information from vertex logs includes the I/O rate from each Dryad channel. The performance counters include over 80 entries describing

the local and global measurements which include information about the .NET runtime. Remote execution daemons provide information about all the Dryad channels.

The filtered data from all the sources is extracted and aggregated in the database accessible from all the local workstations. Data schema is the most important part of the database. The data collection process generates this data schema. The schema categorizes each table column as: numerical data, category, string, timestamp or time offset.

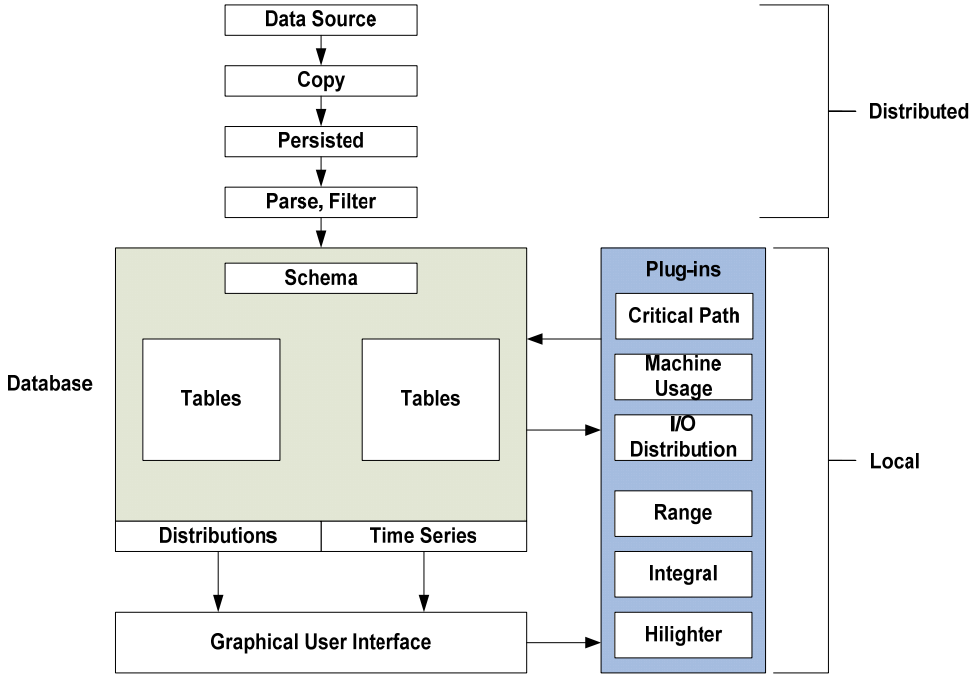

Figure 3.7: Artemis architecture [5]

The data visualization part of Artemis is not tied to the data semantics and is completely generic. Two types of magnitudes are mainly displayed [5]:

- Scalar valued measurements
- Time series data

The data analysis part of Artemis is performed with plug-ins. The plug-ins are used for: for performing application specific data analysis including running scripts invoking other plug-ins, computing meaningful features from the data for driving statistical analysis and for invoking statistical analysis.

A library of generic plug-ins for computing simple data transformations which can be transformed to compute new metrics from existing data is provided. The plug-ins can be invoked as batch computations from other plug-ins or interactively from the GUI [5]. The data that is generated by the plug-ins is merged into the database for further analysis or immediate visualization. While analyzing Dryad jobs the computed features are combined with the scalar metrics extracted from the logs to generate new features for each Dryad process. Statistical machine learning plug-ins can use these features.

Recently, plug-ins interfacing Artemis with two off-the-shelf statistics packages has been implemented: feature selection and data clustering and statistical pattern classification. Data is written to files as expected by each analysis package by the plug-ins and then invoke external processes to perform analysis.

The clustering analysis is used to find the differences in performance among the analyzed entities and also to determine which of the features correlate with those differences. The pattern classification analysis aims at understanding the differences in the performance of the vertices belonging to the different clusters. A model is automatically induced that discriminates between the vertices according to their cluster.

The entire measurement database is accessed by the most powerful kinds of Artemis plug-ins which can implement complex data analysis by joining information from multiple tables

in an application specific way. Three examples of plug-ins developed specifically for analyzing Dryad job have been given below [5]:

- **Machine usage:** It is used to compute machine usage for an entire job and represents it using time series data while enabling visualization with the existing GUI.
- **Critical path:** It shows the longest chain of dependent processes in a Dryad job graph execution.
- **Network utilization:** It is used to combine various database tables to compute the network traffic distribution.

Artemis is useful for executing long term studies of statistical properties on large scale systems and also as a tool for diagnosing the performance of running an algorithm on a large cluster of computers. Artemis was built as an extensible tool which addresses end to end process of distributed log analysis including; computing, correlating, preprocessing and extracting features, collecting, persisting and cleaning the raw logs and visualizing analyzing the data. Artemis utilizes techniques such as distributed computing, databases, visualization and statistical analysis and machine learning.

Providing the end user with automatic application specific diagnosis of performance problems is the end goal of this research. Artemis can handle raw data management, feature extraction and data summarization and statistical analysis.

4. PICVIZ

Picviz is free software and can be downloaded from http://wallinfire.net/picviz/ [1] for further study. Picviz is available in two versions: one for Windows machine and the other for Linux machine. Both the versions can be downloaded from the website mentioned above. Picviz [2] implements the use of parallel coordinates to display an infinite number of events in multiple dimensions thus, creating a parallel coordinates image. Relevant computer security information can be extracted from logs using this generated graph. It can be used to discover attacks, software malfunctions and other high level issues related to the system. Picviz helps us in better understanding, finding network anomalies and seeing how unexpected things can be discovered.

The log files cannot be directly given as input into the Picviz. Therefore the log files are first converted into the PGDL format [2] before Picviz treatment. CSV a common format to read and write data into log files is converted into PGDL format by using a program file csv2picviz.pl.

Picviz transforms the acquired data into parallel coordinate plot image to view the data and discover interesting results from the image. Visualization makes it easier to analyze a massive amount of log data. Parallel coordinates help to organize them and see the correlations and issues in between the data by looking at the generated image.

The Picviz architecture is shown in figure 4.1 [1] and it consists of the following sections as mentioned below to analyze the graphical image [2]:

- Generation of Picviz (PCV) language
- Understanding the graph
    - Graphical frontend

- Command line interface
- Filtering
- String positioning
- Correlations

As mentioned earlier log data is collected from a variety of sources and converted into PCV language which is used by Picviz to generate the coordinate image. The data is then parsed and rendering plugins and output plugins are applied to the data to generate the output image.

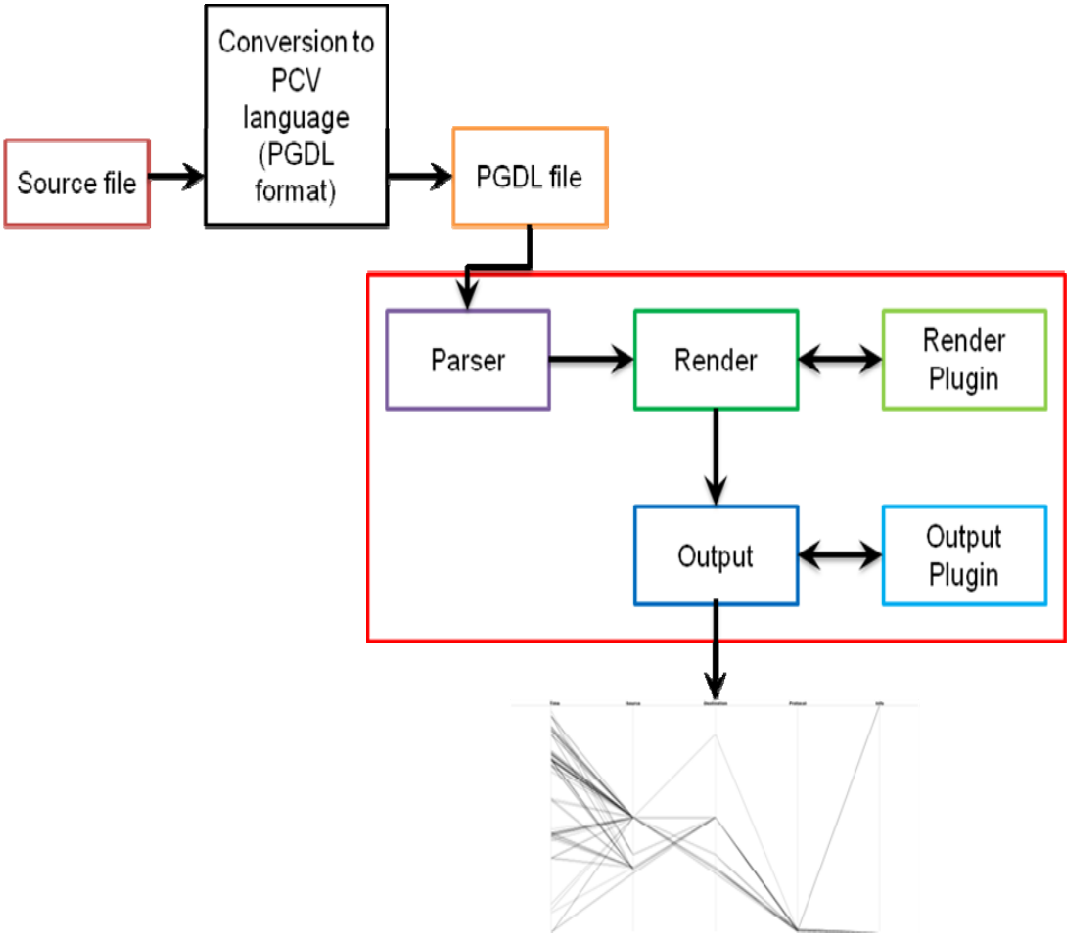

Figure 4.1: Picviz Architecture [3]

Picviz basically consist of three parts, each facilitating the creation, tuning and interaction of parallel coordinates graph [2]:

- Data acquisition
- Console program
- Graphical frontend

In the data acquisition part the log parsers and enricher are used. The console program part transforms the data present in the PGDL file [2] into png or svg image and the graphical frontend transforms the data into a graphical canvas for user interaction as shown in figure 4.3. Each parallel line in the generated graphs strictly defines a single variable as seen in figure 4.3.

4.1 Data Acquisition

Data or log files are collected from apache logs, tcpdump, syslog, network analyzer (Wireshark [6], Kismet [7]), SSH connections, etc. These data files first need to be converted into CSV format. The files in CSV format can be converted into PGDL files using csv2picviz.pl program which is written in perl. This file consists of data in PCV language. This data is then represented visually by the console program or the graphical user interface. This PCV language is divided into four sections [1]:

- header (optional)
- engine (optional)
- axis
- data

The data below represents a typical PGDL file that is generated using csv2picviz.pl which is included in the Picviz source file. The source file also includes Picvizlogparser.py which can also used to generate the PGDL file.

The log variable consists of strings of variable forms depending on the application writing it and what information it is trying to represent which makes the logs disorganized [2]. This string can be used to extract other more useful variables such as the IP address, username, success or failure and so on.

```
header {
   title = "Unicorn";
}
axes {
   timeline axis0 [label = "Time"];
   ipv4 axis1 [label = "Source"];
   ipv4 axis2 [label = "Destination"];
   port axis3 [label = "Sport"];
   port axis4 [label = "Dport"];
   5 axis5 [label = "Spkts"];
   5 axis6 [label = "Dpkts"];
   250 axis7 [label = "Sbytes"];
   250 axis8 [label = "Dbytes"];
}
data {
 axis0="22:34:21.658559",       axis1="192.168.1.160",       axis2="192.168.1.158",
axis3="51312",  axis4="22273",  axis5="1",  axis6="1",  axis7="78",  axis8="54";
 axis0="22:34:21.661383",       axis1="192.168.1.160",       axis2="192.168.1.158",
axis3="8602",  axis4="3801",  axis5="1",  axis6="1",  axis7="78",  axis8="54";
 axis0="22:34:21.665039",       axis1="192.168.1.160",       axis2="192.168.1.158",
axis3="10154",  axis4="6668",  axis5="1",  axis6="1",  axis7="78",  axis8="54";
 axis0="22:34:21.668108",       axis1="192.168.1.160",       axis2="192.168.1.158",
axis3="62283",  axis4="123",  axis5="1",  axis6="1",  axis7="78",  axis8="54";
}
```

Figure 4.2: PGDL source file

4.2 Console Program

This program is used to generate static png and svg images. The command used to convert a PGDL file into a png or svg image is as given below:

$ pcv –Tpngcairo file.pgdl –o file1.png

The program file for this operation is present in picviz-cli-0.6 zip file under the name ‘pcv’. In order to generate the static image the user has to enter certain other information along with the input and output file name. The user needs to mention the output file format if it should be in png, svg or text format. The information contained in the image can be manipulated by use of various plugins. The list of various plugins that are currently available are given in table 4.1 below [3]:

Initials	Description
A	Gives the argument to the plugin
a	Displays text
d	Starts in debug mode
L	Draw the text every N lines
l	We don't learn the string algo
m	Add text labels only on minimum and maximum plotted values
o	Output to a file instead of stdout
p	Sets the PID file to use for real-time mode
q	Quite mode on
r	Increases the image resolution
R	Rendering plugin (heatline, …)
S	Listen to the specified socket
T	Output plugin (pngcairo, svg, …)
t	Template file to use when run in daemon mode
V	Picviz version

Table 4.1: List of available plugins [3]

Changes have been made in this pcv file [3] during my research to include two new plugins ‘E’ and ‘e’ which when used groups’ data lines and provides a better and clearer image. These plugins are described in detail later on in this paper. Log can consist of millions of data line which is difficult to handle by the GUI that is why a static image generated by the console program is more useful. When only thousands of data lines are present in the log or only a part of log needs to be viewed GUI is useful.

4.3 Graphical User Interface

The graphical user interface is as shown in figure 4.3. It can be used to interact with the log data. The position of the parallel lines can be interchanged or removed. It can be used to select and visualize a single data line. Each line can also be colored as required [1]. The distance between two parallel lines can be changed. The best part about this GUI is that the data line generation over the entire time period can be viewed step-by-step. The difficulty is that data cannot be filtered and it is difficult to generate image of millions of data line. Also since the GUI currently available is the beta version it is slow and crashes a lot. The GUI is coded in python [1].

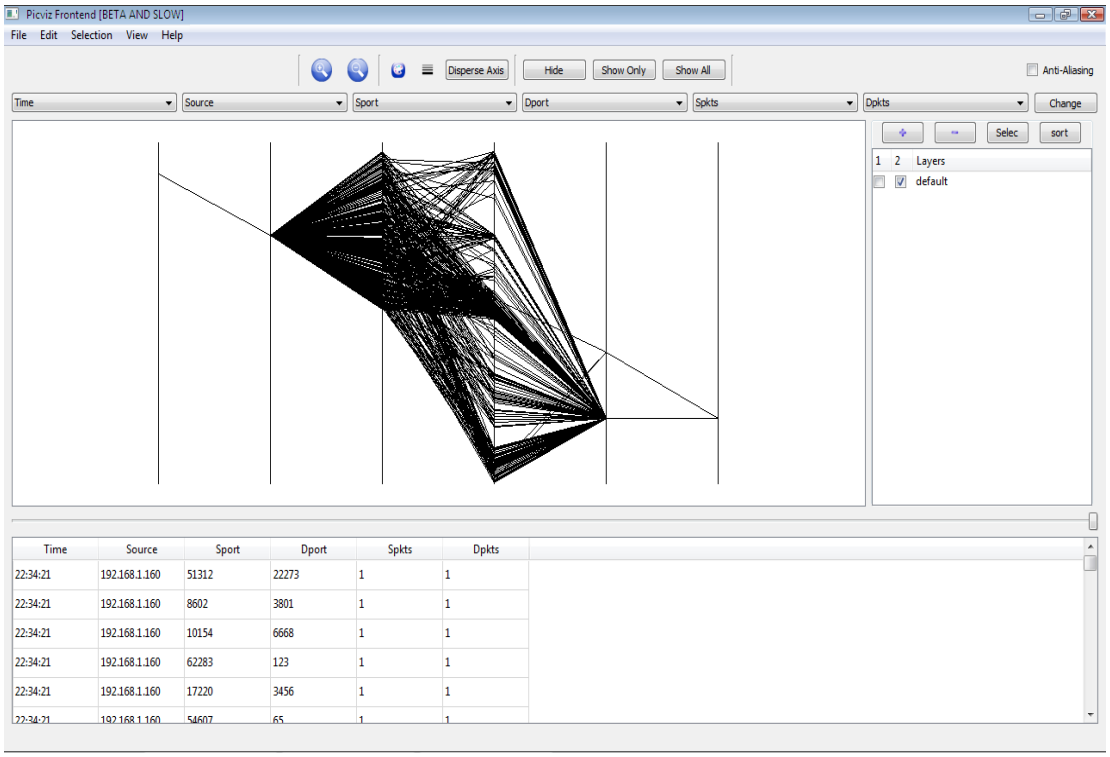

Figure 4.3: Picviz GUI

5. CHALLEGES IN PICVIZ

Picviz software faces many challenges since it is just the beta version of the software. The console interface is better than the graphical interface since it is easier to deal with a static image as compared to a dynamic image. Also, as mentioned earlier the amount of log data also plays an important role in selecting the type of interface used, since console interface can handle millions of lines of log data but it is difficult for the graphical interface to handle this much data. It is better to use the graphical interface only thousands of lines of log data are present. The graphical interface also provides enhanced interaction facilities with the log data. One drawback of the graphical interface (GUI) is that the heatline plugin that is present with the console interface (CLI) is not present and also the filter plugin is not present. The static image generated by the CLI also has certain challenges related with it. One of the challenges is that image can be manipulated only through the command line and also some of the feature that are present in GUI is not present in the CLI, for example Layers, Brushing, Undo/Redo, Increase line width, etc.

The main challenge present in the static image generated by the CLI is of clustered and overlapping data lines. The figure 5.1 below represents one such static image generated using the Picviz. This image consists of 110375 data lines without any labels. It can be seen that it will be very difficult to understand such an image. Also it is impossible to select a single line and view its information. All the lines are clustered together and there are parts of the image that are completely black and a single line is invisible. It is difficult to extract any kind of information from such a dense image. Adding n more lines will make the image even denser. The image below does not contain any labels and it is still so difficult to understand it. If text is displayed for each line, the text also becomes unreadable since the text also overlaps the image. So in the

end only dark black patches are seen in the image and no useful information can be extracted from this static image.

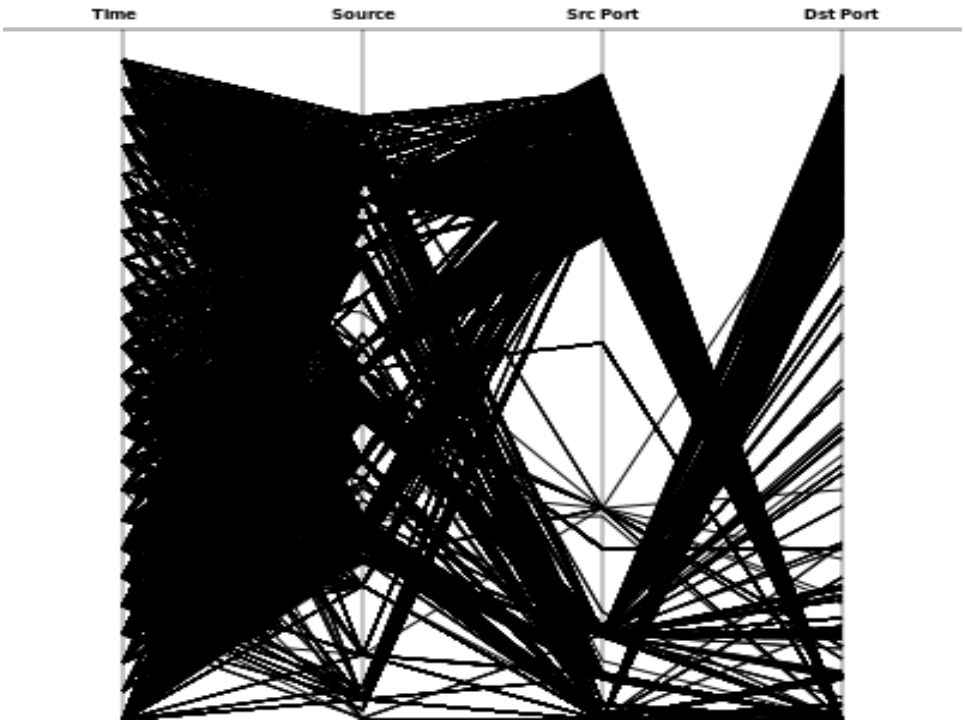

Figure 5.1: Static image generated by Picviz

Using heatline plugin does not help since all the lines are still generated and it becomes difficult to view the information of a single line. The heatline plugin provides a color mask for the image based on event frequency where green is for the lowest frequency, red is for the highest frequency and yellow is for medium frequency. The filter plugin can be used to view particular sets of data line for example, IPaddress = 190.160.35.1 or for Protocol != TCP. In the first case all the data line that consist of IPaddress = 190.160.35.1 are present in the static image and rest are deleted and in second case all the data line that consist of Protocol = TCP are deleted from the static image and the rest of the line are present. The problem with this is that only part

of the log information is present in the image. The entire information present in the log data is not represented in the image. The heatline plugin and filter plugin present in Picviz try to help us in understanding the image and discover certain hidden information from the log data. But still these two plugins cannot help us understand the log data in its entirety.

The static image is such that not much manipulation can be done on it and it is difficult to select a single line from the static image as can be done in the GUI. There is no way of compressing the entire data in such a way the entire information is still available but the image is simplified so that it is much easier to read and understand the data. A method is required that can compress the entire data and reduce the number of data lines generated in the static image while still containing most of the information that is present in the original static image.

6. PROPOSED SOLUTION

Picviz [1] software faces the clustering and overlapping problem as described in the earlier section. This section provides solution for this problem. A way to overcome this problem is to group data into different sets and to represent each data set as a single data line in the static image. There are a number of ways in which grouping of log data can be done two of which are described below. These two methods have been identified as providing the maximum amount of the original information while still reducing the number of data lines in the static image. The two methods of grouping are as follows:

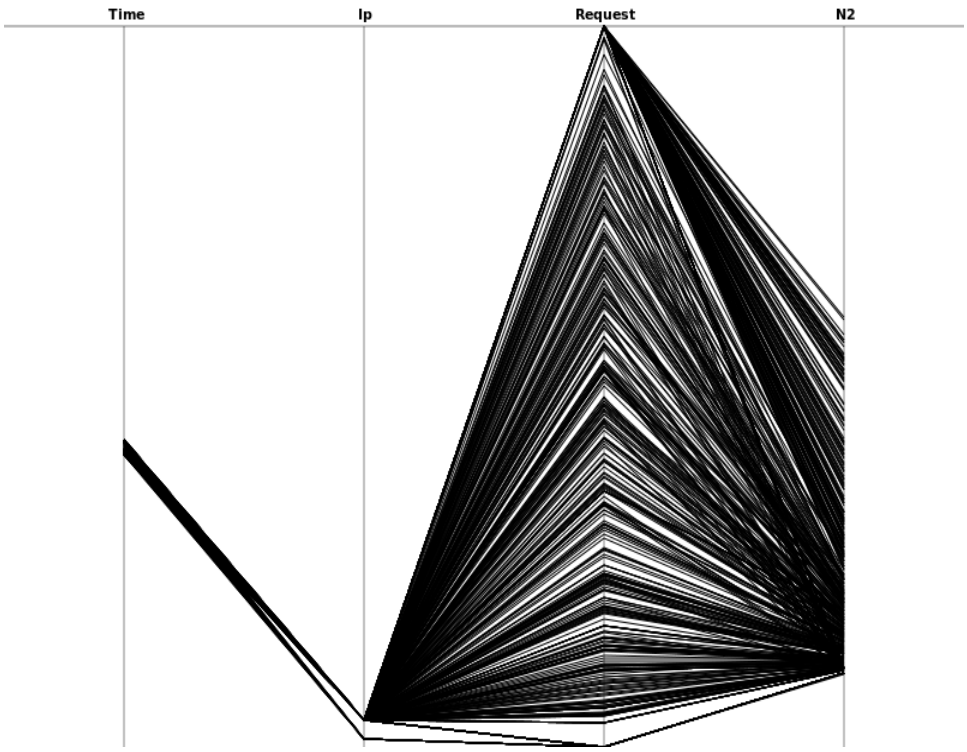

Figure 6.1: PCV image generated before frequency grouping

6.1 Grouping based on Comparison of data (-E plugin)

In this type of grouping the values of each data line is compared with the values of its consecutive data lines and if the values are same then they are grouped together. The values of each data variable for a line are compared with the value of the same data variable but for its consecutive lines. To simplify this, the value of IP variable for data line 1 will be compared with the value of IP variable for data line 2, 3, 4,… Same goes for other data variables which may be present in the log file. Only the value of the Time data variable will not be compared. This is because it will be used to calculate the average time for all the data lines of a particular set and then display it depending on the average time value in the parallel co-ordinate plot.

The data line values for each data variable is in string format and represented in the parallel co-ordinate plot in the same format. The Time variable is the only data variable that is converted into integer and then placed on the plot based on its integer value. If the text label plugin is used it can be seen that the time value increases along the y axes, that is the data line with the oldest time stamp is represented first then the one with the second oldest time stamp and so on. This type of line by line generation can be viewed in Picviz GUI. So inorder to find the proper place in the plot to represent a particular set the average of the time values is calculated and then based on the calculated value it is represented in the plot. After calculating the average time, the position of the data line may be same as one of the older data lines or may be completely different in any way only a single line will be generated for the entire set which is the main objective.

In this type of grouping based on comparison values only of a particular data variable will be compared which has the most importance such as either for IP, Protocol, etc. This will be useful if we require information regarding all the IP address but it is not important which

protocol they are using or which URL they are visiting. As seen in figure 6.1 there is a cluster in the data variable Request and N2. In this case the data variable values for N2 are compared since the value of Request is of not much importance. It was seen that because of such kind of grouping the number of data line generated was considerably reduced. This made the image less dense and it was easily understandable.

This method is somewhat similar to using the filter plugin but the advantage is that all the values for the required data variable are represented in the static image. The missing lines are of the data variable that has not much importance while finding an attack or anomaly. This type of grouping will be more beneficial when attacks or anomaly for a particular data variable is being studied. This type of grouping provides more information about a particular data variable that is being studied. It also contains more information as compared to the second type of grouping described below.

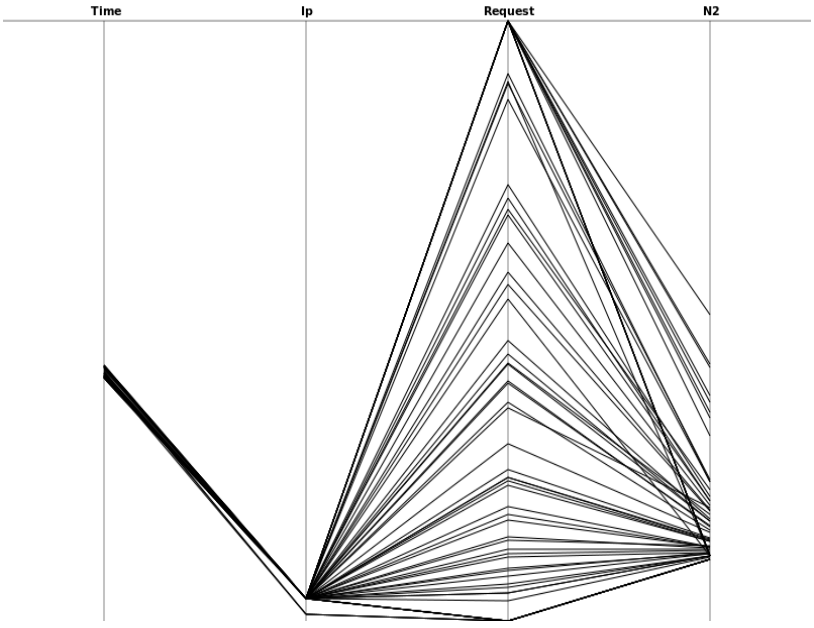

Figure 6.2: PCV image generated after grouping based on comparison of data

A new plugin value'-E' has been introduced to perform this kind of grouping. The name of one of the axes that is to be used for comparison is passed along with the –E plugin. All the values for that axes are compared and the values that are same are combined to form a single line. The format of the command to use this plugin is given below:

$ pcv –Tpngcairo –EIp file.pgdl –o file1.png

In the above command E represents the –E plugin and Ip represents the name of the axes whose data needs to be compared. The name of the axes change based on the requirements of the user.

6.2 Grouping Consecutive data (-e plugin)

The data lines for this technique are grouped together based on its consecutive nature. This means that 10, 20, 50, …. data lines are grouped together to form a single set and as mentioned earlier each set is described by a single line. This is useful since log data is generated for every second and sometimes even for a fraction of second. In such cases there is not much change in the values of the data variables. Such data lines can be combined together to form a single data line.

In this type of grouping more information as compared to the previous technique may be lost since some of the information contained in the data lines may be deleted. While performing such kind of grouping the number of data lines that should be grouped together should be specified. Based on this number the density of the image will differ. The number should be selected based on the size of the log file since huge log file will require a bigger number and smaller log files will require smaller number to obtain an appropriate image. The number of grouped data lines can be varied for each log data file to obtain useful information. This needs to

be done since some information may be present in one image of the data file and absent in another image of the same data file.

The average time value for each line is calculated for this technique also because of similar reasons as mentioned in the previous technique. This is done to mainly find the position of the of the data line in the y axes. The time values of each data line present in a set are added together and divided by the total number of lines that are present in the set.

This method has been extended to include log files which contains data from a couple of months or years. In such cases the data lines generated represent a particular month during which they were generated or even a particular year. All the data lines generated in a particular month are combined together at one point on the y axes. Multiple lines can be generated from that point on the y axes but they are all combined together at that one point on the timeline axes. The image generated from such kind of grouping is shown below in the results section. This kind of grouping is explained in more detail in the same section. This is an extension of the Grouping based on Consecutive data method.

A new plugin value'-e' has been introduced to perform this kind of grouping. The number of data lines that are to be combined together are given along with the plugin. The format of the command to use this plugin is given below:

$ pcv –Tpngcairo –e50 file.pgdl –o file1.png

In the above command e represents the –e plugin and 50 represents the number of data lines that will be combined together to form a single line. This number can be changed based on the user requirements and to improve the clarity of the image.

Figure 6.3 below has been generated when this plugin has been used. The difference between figure 6.1 and 6.3 is easily visible. It is quite easy to read and understand the

information present in figures 6.2 and 6.3. There is also a slight difference in figure 6.2 and 6.3, it can be noticed that the second IP address value is present in figure 6.2 whereas it is absent in figure 6.3. Therefore, it has been mentioned that more information is present in the image when the first technique is used.

Changes have been made in the pcv.c, render.c, correlation.c, engine.c, engine.h and render.h files. The grouping of data is done in the render.c file since it generates the information about which line should be generated and where in the image. The average time value is also calculated in this file. The pcv.c and engine.c files are manipulated to add the new plugin into the software. Correlation.c is manipulated to use the heatline plugin along with the new frequency plugin. This new plugin has been introduced only in the console program and not in Picviz GUI.

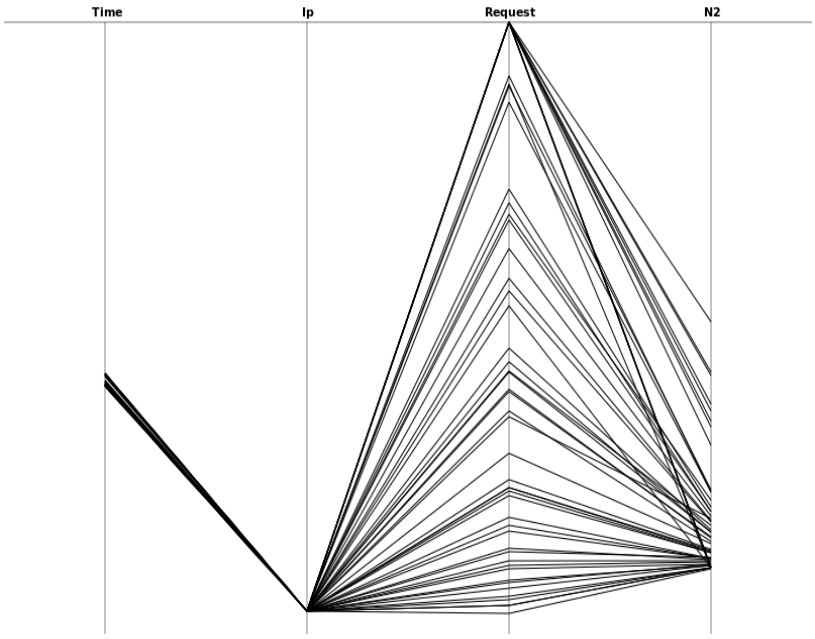

Figure 6.3: PCV image generated after grouping based on consecutive data

The data has been collected using Wireshark [6] which is a network analyzer. It records all the transactions done by the system with the wireless router. It stores time, protocol, source and destination packets and list of websites visited. The advantage of using this software to generate log data is that the collected data is already in CSV format and ready to be converted into PGDL format. Data has also been collected from event logs to show the power of using second method of grouping in case the log data spans a period of several months or even years.

7. RESULTS, OBSERVATIONS AND DISCUSSION

This section gives a detailed description of the images that were generated from using the above mentioned plugins and also explains the importance of such plugins during log data analysis. The obtained results are explained and shown below and the advantages of these two plugins based on these individual results are also explained.

7.1 Test case 1

One of the problems with the generated images is the presence multiple lines generated from a single point. The use of –E plugin solves this problem. It combines all the lines generated from a single point on the y axes into a single line. This means that if the destination axes consists of 10 different IP addresses and 1000 data lines are generated containing these 10 different IP address, then what –E plugin does is reduce these 1000 data lines into just 10 data lines. These plugins don't affect the actual log data, it just affect the lines that are generated in the image.

Figure 7.3 consists of 6 different destination IP addresses but it can be seen that more than six data lines are generated from these IP addresses. –E plugin reduces the number of lines and generates figure 7.4 containing only 6 different lines between Destination axes and Protocol axes. As can be seen from both the figures the rest of the lines that are generated between Time axes and Source axes and also between Source axes and Destination axes remain the same. The only lines that are affected are between Destination axes and Protocol axes. As explained in earlier section that, using the –E plugin affects the data lines between the mentioned axes and the axes immediately following the mentioned axes.

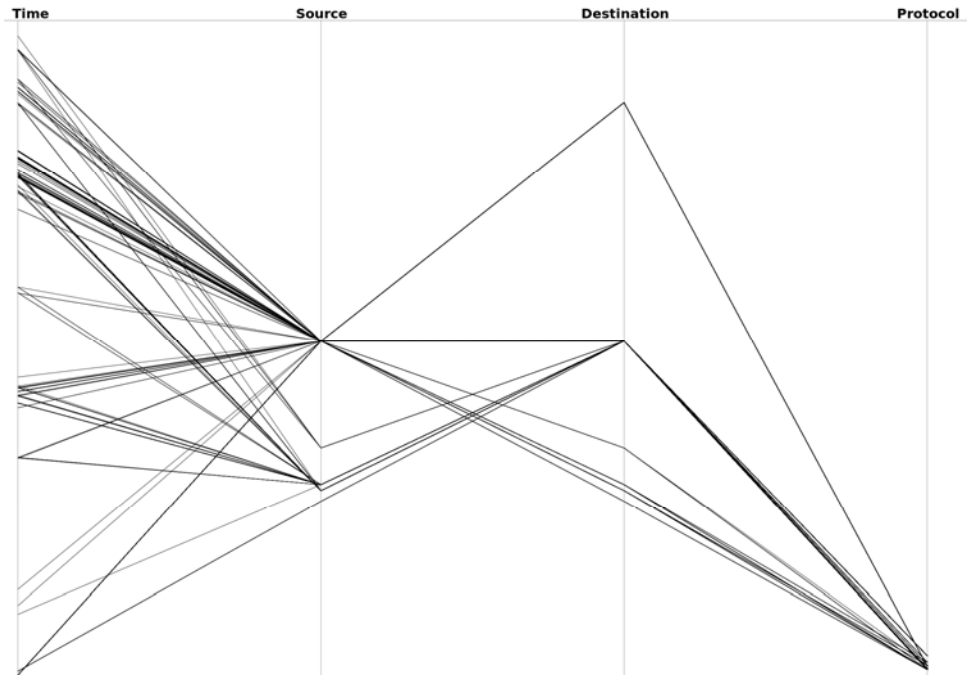

Figure 7.1: Image generated before using –E plugin

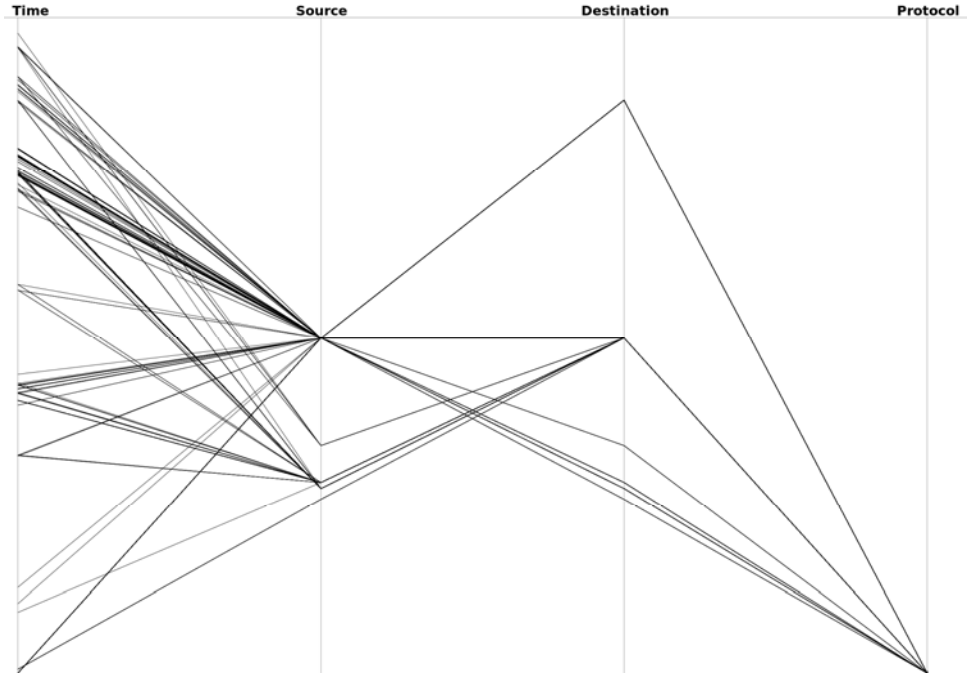

Figure 7.2: Image generated after using –E plugin

7.2 Test case 2

The figure 7.1 is generated before using –e plugin. In this figure it is impossible to read the values that are present in the figure. It contains too many things in a small space and thus, it becomes difficult to understand anything from the generated image. It just looks like a bunch of black lines that does not make any sense to the viewer. It is difficult to read the IP address present at the destination axes. Also many of the values present on the source port axes and destination port axes are difficult to read since there is lot of overlapping data that is present. In figure 7.2 which is generated after using –e plugin the IP address at the destination axes is clearly visible and also some of the port numbers can be clearly seen. It can be seen that the minimum source port number is 4100.

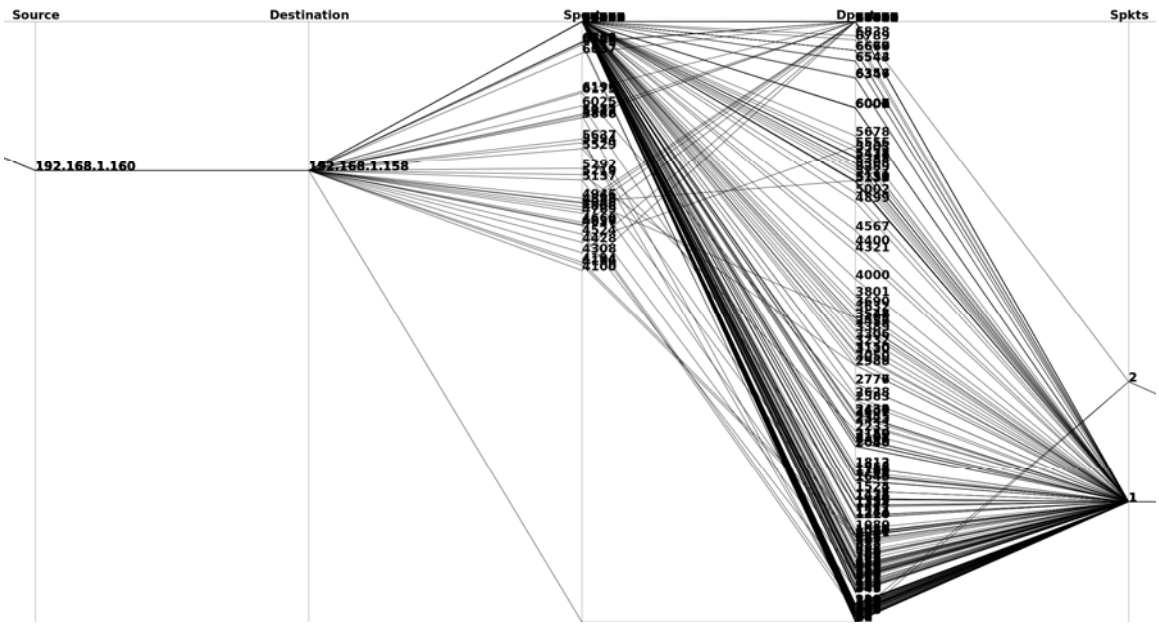

Figure 7.3: Image generated before using –e plugin

The main objective of this software is to ease the task of log analysis by representing it visually but it is not much helpful if the values present in the generated image is impossible to

read. Using the above mentioned plugins solves this main problem and makes the image and the values present in it easily readable and understandable. Using this plugin will not only help us understand the log data but also save some time in trying to figure out the maximum or minimum values, checking for security breaches, etc.

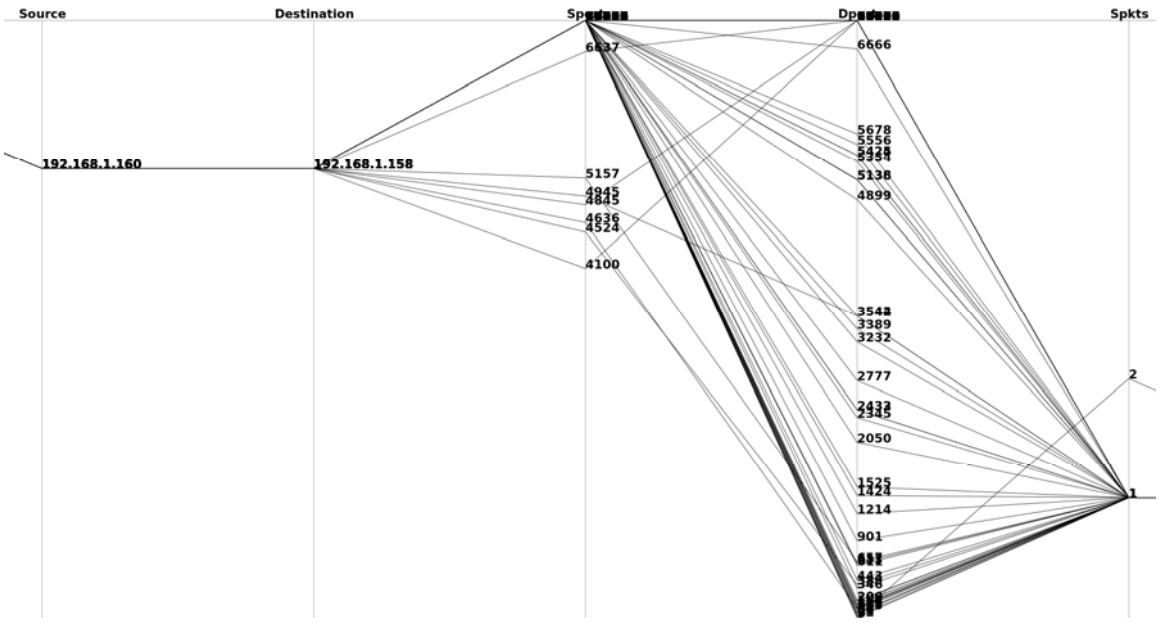

Figure 7.4: Image generated after using –e plugin

7.3 Test case 3

This is an additional feature that has been included in the –e plugin. This feature is activated automatically in case the log data consists of data lines that span a long period of time such as couple of months or even years. The figure 7.5 was generated from systems log that was collected over a period of 12 months. In this case all the data lines that were generated over a period of one month have been combined together to form a single line. Each line generated from the Time axes represents the data generated during that one entire month. In case the data spans a period more than 5 years then each generated data line in the figure represents data lines

generated over a period of one year. In case the data spans a period equal to or less than 5 years then each generated data line in the figure represents data lines generated during a single month. Hence in the generated image the month and the year is clearly visible but not the date and time when it was generated. This feature greatly reduces the number of lines present in the image since it is grouped based on the month or year they were generated. The rest of data points generated on the various other axes is based on the frequency of the occurrence of that data point. The data point with the highest frequency for that month or year is represented in the generated image. Like in the figure below Event ID 4908 had the highest frequency for a particular month so it is present in the figure. Similarly, Event ID 1108 had the highest frequency for other month so it is also present in the figure. In the Task Category axes Special Logon, Logoff, Logon, etc were the data points with highest frequency for a particular month so they all are present in the figure. This process of data point generation is used for all other axes except the Date and Time axes.

In such cases were data spans over a long period of time, –e plugin overrides the process of combining the number of data lines specified by the user. The user can give any number as input along with the –e plugin but the image generated is still not affected, for each value the same image is generated. This is additional feature that is included into the coding of –e plugin to generated a more clear and readable image. In this case some of the data point for other axes are lost since only the data point with the highest frequency are generated in the image but still it gives a better overall view of the generated data which was earlier not possible.

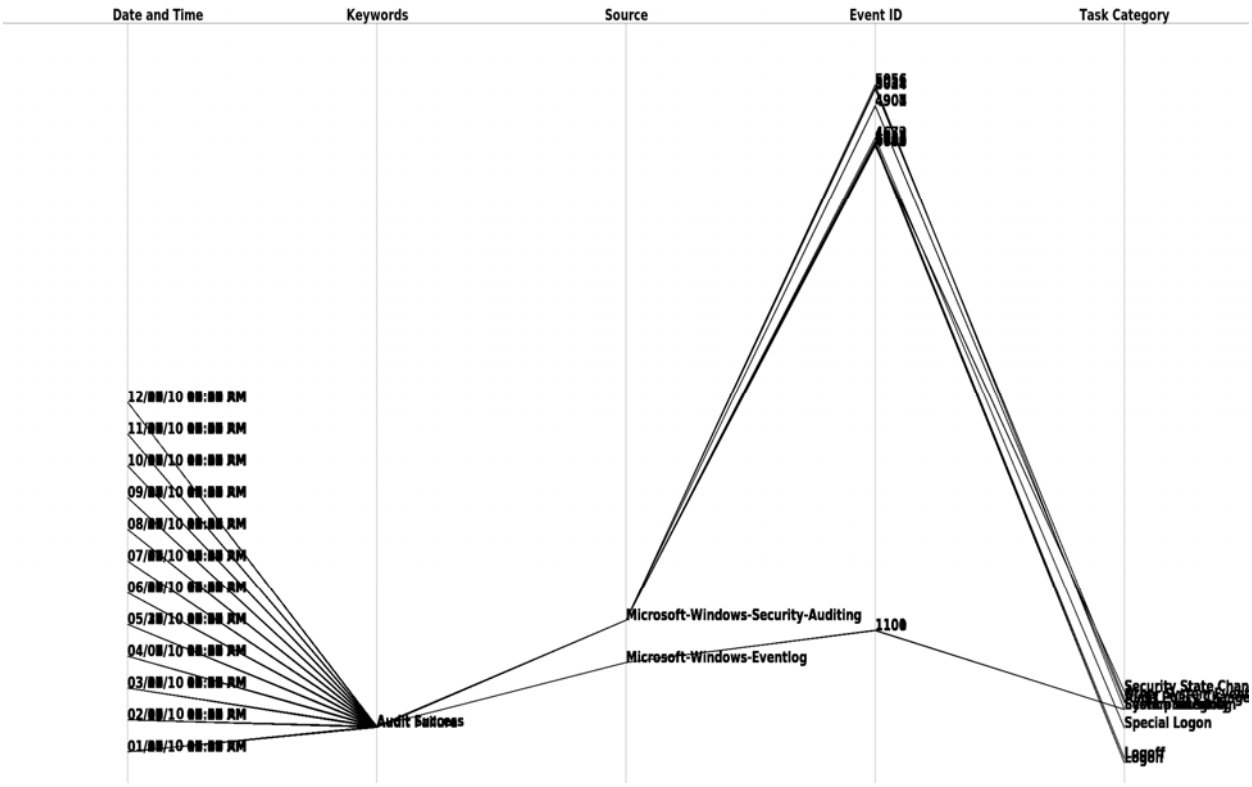

Figure 7.5: Image generated using –e plugin when data was collected over 12 month period

7.4 Test case 4

The Picviz software comes with a built in plugin called as heatline. This plugin color codes the generated image based on the frequency a line is drawn between two axes. This means that say destination ‘192.160.45.2’ uses protocol ‘TCP’ significantly more than protocol ‘UDP’ then the line generated between ‘192.160.45.2’ and ‘TCP’ will be colored red and the line generated between ‘192.160.45.2’ and ‘UDP’ will be colored green when the heatline plugin is used. This color coding is based on how frequently a line is drawn between the same data points on two axes. The heatline mode breaks lines color to create a gradient from green (low frequency event) to red (highest frequency event) via yellow (medium). The figure 7.6 given below is generated

when the heatline mode is used to generate the image. The format to use this plugin is as given below:

$ pcv –Tpngcairo file.pgdl –o file1.png –Rheatline

When the heatline plugin is used along with the –E plugin or –e plugin, it proves very useful for data analysis. As seen from figure 7.6 that using heatline plugin does not help much with the clustered line problem, it only makes the image more colorful but it is still difficult to read any kind of written data in the image.

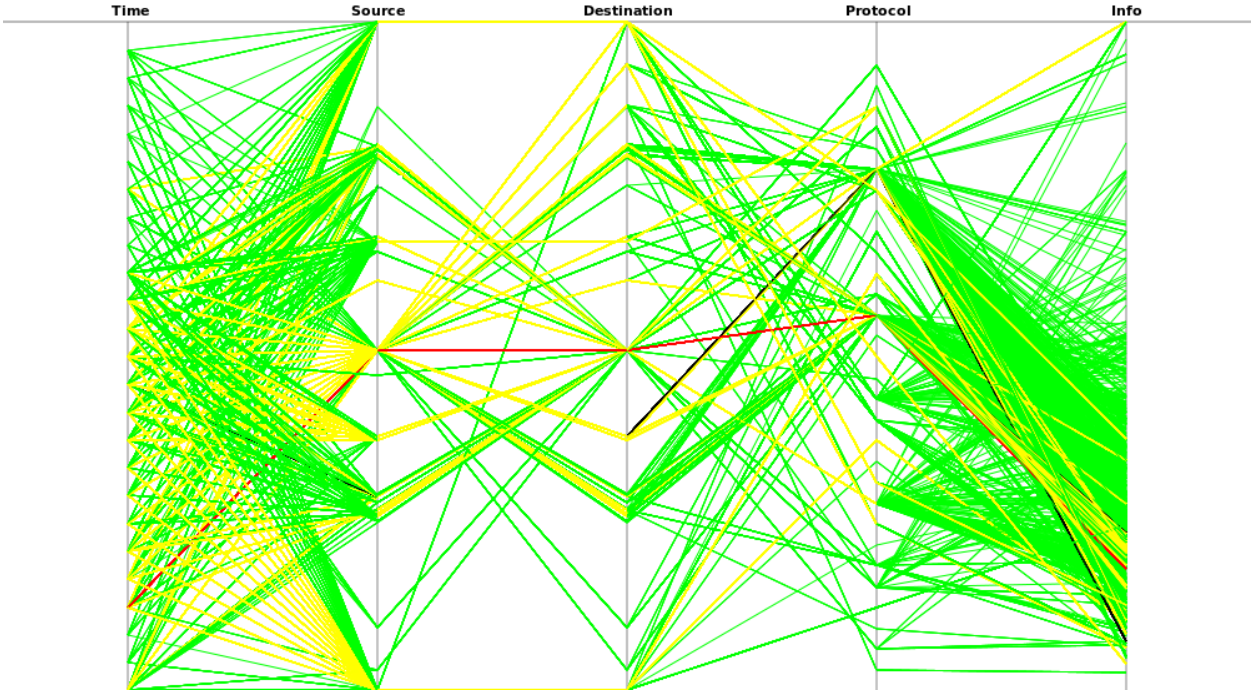

Figure 7.6: Image generated when using heatline plugin

When –e plugin is used along with heatline plugin figure 7.7 is generated. It can be seen that figure 7.7 consists mostly of lines colored yellow and all the lines colored red. This means that the same data line is repeated a lot of times in the log data and such repeated data line is displayed in the image. This helps us understand the frequency of particular data points. In such

cases the disadvantage of using –e plugin is that we always know that the data lines with less frequency is surely deleted. This feature is useful to study the usage patterns in the network, like finding the user who uses the most amount of internet band width, or the website that is more frequently visited, or the protocol that is generally used. This is also useful to find the time when there is maximum usage of the internet. It can be seen in figure 7.7 can answer such network usage questions if values are also present in the image. Also since the number of generated lines is reduced it will be easy to read the generated data.

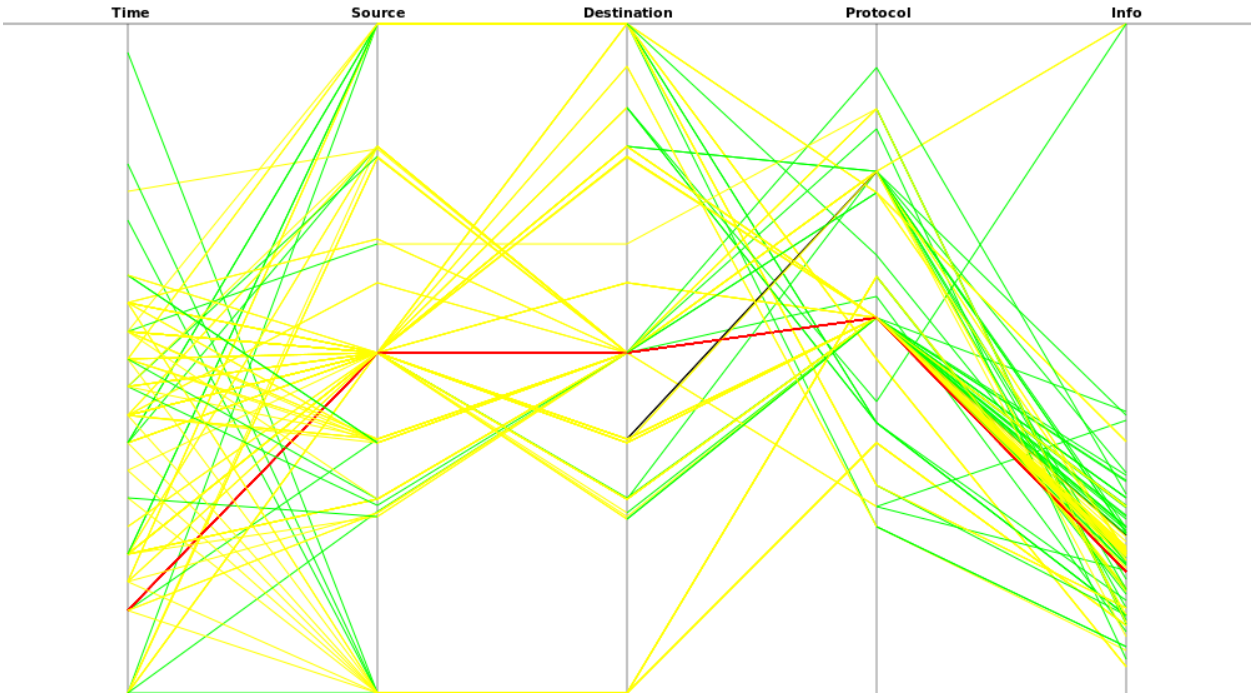

Figure 7.7: Image generated after using –e plugin along with heatline plugin

The above test cases prove that the use of –E plugin and –e plugin can significantly improve the readability of the image. It also helps in easing the process of log data analysis of huge amounts data which was earlier difficult because of the congested line problem.

8. FUTURE WORK AND CONCLUSION

Future work will be to find other more efficient ways of grouping the data so that no information is lost in the process. Another future work will be to incorporate these new frequency plugin as well as filter plugin into the Picviz GUI [1]. Although it is possible to color any line in the Picviz GUI but the heatline plugin is not present. This plugin needs to be included in the GUI.

Since the log file consists of thousands and millions of data it becomes difficult to represent the entire thing in a single image. The image becomes dense and it becomes difficult to process the image and obtain information from it. The advantage of these new grouping techniques is that it reduces the number of lines that are generated by the image considerably while still retaining most of the information present in the log data. Two new plugins –E and –e have been introduced in the Picviz software to facilitate the use of these two techniques. The image becomes clearer and easily readable. The primary goal is to ease the analysis of data and finding correlation among the various variables and using these plugins enables us to do so.

REFERENCES

[1] Picviz. http://wallinfire.net/picviz/, 2009.

[2] S. Tricaud, "Picviz: finding a needle in a haystack," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[3] S. Tricaud and V. Amaducci, "Know your tool: use Picviz to find attacks," 2007.

[4] Riska and E.Riedel, "Data Drive Workload Captured in Logs Collected During the Field Return Incoming Test," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[5] G. Cretu-Ciocarlie, M. Budiu and M. Goldszmidt, "Hunting for problems with Artemis," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[6] Wireshark. http://www.wireshark.org/, 2010.

[7] Kismet. http://www.kismetwireless.net/, 2010.

[8] J. Tan, X. Pan, S. Kavulya, R. Gandhi and P Narasimhan, "SALSA: Analyzing Logs as State Machine," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[9] F. Salfner and S. Tschirpke, "Error Log Processing for Accurate Failure Prediction," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[10] C. DiFatta, M. Poepping, D. V. Klein, "Carnegie Mellon's CyDAT: Harnessing a Wide Array of Telemetry Data to Enhance Distributed System Diagnostics," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[11] E. W. Fulp, G. A. Fink and J. N. Haack, "Predicting Computer System Failures Using Support Vector Machines," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[12] S. Sandeep, M. Swapna, T. Niranjan, S. Susarla and S. Nandi, "CLUEBOX: A Performance Log Analyzer for Automated Troubleshooting," in *Proc. of the USENIX Workshop on Analysis of System Logs* (WASL), San Diego, 2008.

[13] Hadoop, 2007: http://hadoop.apache.org/core.

[14] J. Dean, S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *USENIX Symposium on Operating Systems Design and Implementation* (OSDI), pp. 137-150, San Francisco, 2004.

[15] M. Isard, M. Budui, Y. Yu, A. Birrell, D. Fetterly, “Dryad: Distributed data parallel programs from sequential building blocks,” in *European Conference on Computer Systems* (EuroSys), Lisbon, Portugal, 2007.

[16] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, J. Currey, “DryadLINQ; A system for general purpose distributed data- parallel computing using a high-level language,” in *Symposium on Operating System Design and Implementation* (OSDI), San Diego, 2008.

[17] G. Conti and K. Abdullah, “Passive visual fingerprinting of network attack tools,” in *Proc. of the ACM workshop on Visualization and data mining for computer security*, pp. 45–54, New York, 2004.

[18] E. R. Gansner, E. Koutsofios, S. C. North and K. Phong Vo, “A technique for drawing directed graphs,” in *IEEE Transactions on Software Engineering*, vol.19 no. 3, pp. 214–230, March 1993.

[19] A. Inselberg and B. Dimsdale, “Parallel coordinates: a tool for visualizing multi-dimensional geometry,” in *Proc. of the 1st conference on Visualization*, pp. 361–378, San Francisco, 1990.

[20] T. Y. Lin and D. P. Siewiorek, “Error log analysis: statistical modeling and heuristic trend analysis,” in *IEEE Transactions on Realiability*, vol. 39, pp. 419-432, October 1990.

[21] P. Barford, J. Kline, D. Plonka and A. Ron, “A signal analysis of network traffic anomolies,” in *Proc. of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, France, November 2002.

[22] Analog. http://www.analog.cx/, 2004.

[23] Open Web Analytics. http://www.openwebanalytics.com/, 2010.

[24] W3Perl. http://www.w3perl.com/, 2010.

[25] Webalizer. http://www.webalizer.org/, 2009.

[26] AWStats. http://awstats.sourceforge.net/, 2009.

[27] J. P. Rouillard, “Real-time Logfile Analysis Using the Simple Event Correlator (SEC),” in *Proc. of the 18th USENIX conference on System Administration*, pp. 133-149, Atlanta, November 2004.

[28] W. D. Pauw and S. Heisig, “Visual and algorithmic tooling for system trace analysis: a case study,” in *ACM SIGOPS Operating Systems Review*, vol. 44, January 2010.

[29] K. Zhu, K. Fisher and D. Walker, “Incremental learning of system log formats,” in *ACM SIGOPS Operating Systems Review*, vol. 44, January 2010.

[30] S. Wedig and O. Madani, "A Large-Scale Analysis of Query Logs for Assessing Personalization Opportunities," in *Proc. of the 12th ACM SIGKDD*, pp. 742-747, Philadelphia, 2006.

[31] A. Cooper, "A Survey of Query Log Privacy-Enhancing Techniques from a Policy Perspective," in *ACM Trans. Web, 2, 4,* Article 19, 2008.

[32] M. Bendersky and W. B. Croft, "Analysis of Long Queries in a Large Scale Search Log", in *Proc. of the WSCD*, pp. 8-14, Barcelona, Spain, 2009.

[33] H. Zhu, G. Fu, Y. Zhu, R. Jin, K. Lü, and J. Shi, "Dynamic Data Recovery for Databse Systems Based on Fine Grained Transaction Log," in *Proc. of the IDEAS*, pp.249-253, Coimbra, Portugal, 2008.

[34] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics," in *ACM TISSEC*, vol 2, pp.159-176, 1999.

[35] K. P. Joshi, A. Joshi, Y. Yesha, and R. Krishnapuram, "Warehousing and Mining Web Logs," in *Proc. of the 2nd WIDM*, pp. 63-68, Kansas City, 1999.

[36] P. Muth, P. O'Neil, A. Pick, and G. Weikum, "The LHAM Log-Structured History Data Access Method," in *VLDB Journal*, vol. 8, pp. 199-221, 2000.

[37] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," in *ACM TOCS*, vol.10 no.1, pp. 26-52, 1992.

[38] A. J. Oliner and J. Stearley, "What Supercomputers Say: A Study of Five System Logs," in *Proc. of the DSN*, Edinburgh, UK, 2007.

[39] M. Saleh, A. Arasteh, A Sakha, and M. Debbabi, "Forensic Analysis of Logs: Modeling and Verification," in *Knowledge-Based Systems,* vol.20 no.7, pp.671-682, 2007.

[40] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset, "Analyzing System Logs: A New View of What's Important," in *Proc. of the 2nd USENIX Workshop on Tackling computer systems problems with machine learning techniques*, Cambridge, 2007.

[41] T. Park and I. Ra, "Design and Evaluation of a Network Forensic Logging System," in *Proc. of the 3rd International Conference on Convergence and Hybrid Information Technology*, Daejeon, Korea, 2008.

[42] M. Baglioni, U. Ferrara, A. Romei, S. Ruggieri, and F. Turini, "Preprocessing and Mining Web Log Data for Web Personalization." in *Proc. of the 8th Italian Conference on Artificial Intelligence*, v. 2829, pp. 237-249, 2003.

[43] E. Pinheiro, W. D. Weber and L. A. Barroso, "Failure trends in a large disk drive population," In *Proc. of the USENIX Conference on File and Storage Technologies*, pp. 17–29, 2007.

[44] A. J. Oliner and J. Stearley, “Bad Words: Finding Faults in Spirit’s Syslogs,” in *Proc. of the 8th IEEE International Symposium on Cluster Computing and the Grid*, pp. 765-770, Lyon, 2008.