Performance Analysis of a Network using GriFT Monitoring Technique

Priya Kundal
U.I.E.T, P.U.
Chandigarh

Raj Kumari Bhatia
Assistant Professor, U.I.E.T
Chandigarh

ABSTRACT

Fault tolerance is an influential field of concern while working in a grid. Sharing as its primary goal of evolvement, a Grid includes hardware, software, and heterogeneous resources from different organizations spread over large geographical area which would make it a complex behaviour system. With this composite nature grid systems are hard to manage and will result in a faulty system. To over-come this breach of failure a monitoring technique is required that could observe and analyze the performance of the environment and report the existence of faults. This paper presents the design, implementation and evaluation of the monitoring technique called GriFT which is developed using the concept of Grid Monitoring Architecture (GMA). The analysis is done by deploying it in a real laboratory set-up.

Keywords

Grid Computing, Fault Tolerance, GMA, GriFT monitoring technique.

1. INTRODUCTION

In today's modernized world of high speed computing, computers are playing a vital role in our day to-day life. Due to the expensive nature of supercomputers and frequent need of large computations, it was difficult to process the tasks. So the companies came to the conclusion that rather than buying new machines! Why not they could make use of unutilized resources available in the organization. For this the term Grid Computing came into existence.

Grid computing- The term gird computing is often presented as an paradigm similar to an power electric grid, where a variety of resources contribute power into a shared resource "pool" for many consumers to access on an as-needed basis[1]. In context to computing, It is an environment that has simple idea behind its evolvement- providing users with the ability of sharing and transparently accessing the resources in a distributed and heterogeneous environment. It is a rapidly emerging paradigm for a wide area computing. Its main goal is to provide a service-oriented infrastructure to perform against a common goal, to solve a single task and then may disappear. Grid computing enables virtual organizations to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control, omniscience, and an existing trust relationship. The grid computing can be defined as:

"A type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and user's quality-of-service requirements [2]*."*

Or

"*A computational Grid is hardware and a software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities* [3]".

The Grid Systems are very vast to work with. It involves aggregation and sharing of resources at large geographical area. Management of these resources is important as they may be owned by different organizations with their own policies. Because of highly heterogeneity and complex environment, the chance of faults increases. It is not limited to this; the environment and the applications size are increasing dynamically which demands high availability of resources, and mechanism that is highly scalable, adaptable and reliable. In this highly dynamic environment fault tolerance mechanism becomes a necessity.

Fault tolerance- To over-come the breach of failure in grid a fault tolerant technique is required which would help working in grid a success. This property is necessary for the system that works on parallel running applications, since the failure rate grows with the number of processors. Fault tolerance can be defined as the property of the system that will preserve the delivery of expected services despite the presence of faults-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service [4,5].

The fault tolerance has become the key area of research. Till time there is no single system to which we can say "*it is fault free*" or can handle all its faults completely. Grid is dynamic system and the nodes can leave and join voluntarily. To make fault tolerance system a success we must consider:

- How new nodes join and leave the system.
- How the resources are being used in the system.
- How the resources managed and distributed in the system.

While working with these all these factors, we have to consider desirable properties of failure detectors and correctors that are given as follows [6]:

- Completeness: Any kind of failure (process or machine crash) is eventually detected by all normal nodes.

- Accuracy: The probability of false positives is low.
- Consistency: All processes that are not declared as failed obtain consistent failure information including false positives.
- Flexibility: able to handle various types of network settings.
- Detection Latency: The time taken by a system to detect failure must be low.
- Adaptiveness: Systems should bring up with a small amount of manually supplied information about network settings.
- Low overhead. Monitoring should not have a significant impact on the performance of application processes, computers, or networks.

Phases of Fault Tolerance

There are four phases of fault tolerance in which a complete fault tolerant system works [7]:

- **Fault Detection**

Fault detection or monitoring is the process of recognizing that a fault has occurred. It is often required before any recovery procedure can be initiated.

- **Fault Location**

Fault location is the process of determining the location of faults so that an appropriate recovery can be initiated.

- **Fault Containment**

Fault containment is the process of isolating a fault and preventing the effects of that fault from propagating throughout the system.

- **Fault Recovery**

Fault recovery is the process of regaining operational status or remaining operational via reconfiguration even in the presence of faults.

The greatest problem of recovering from failure is the 1st phase of fault tolerance i.e monitoring phase. Discovering a problem over a huge network like grid is a difficult task as it involves many distributed hardware and software components. The more components it involves lesser will be the chances to identify the problem.

The paper is organised as follows: Section II will briefly describe about Monitoring, its benefits and architecture, Section III we will discuss about GriFT monitoring technique, Section IV will discuss its proposed methodology that we followed to develop GriFT and Section V and Section VI will gives the implementation, results and conclusion and the future scope of the technique

2. Monitoring

Grid monitoring can be defined as the measurement and publication of the state of a Grid component at a particular point in time. To be effective, monitoring must be "end-to-end", meaning that all components between the application endpoints must be monitored. This includes software (e.g., applications, services, middleware, operating systems), end-host hardware (e.g., CPUs, disks, memory, network interface), and networks (e.g., routers, switches, or end-to-end paths)[8]. Monitoring is needed to determine the source of the problem. It involves Error detection, performance analysis, performance prediction, scheduling, etc. Monitoring plays a vital role in grid computing to recover from failure by identifying which component failed and why.

By monitoring the system or observing its working, we can analyse the system performance, can perform fault detection, we can identify the bottle neck in the system, or can tune the performance of the system, and it also helps in showing dependability over other components[9].

The GriFT application is grounded on the Grid Monitoring Architecture (GMA) concept that depicts the relationship between producer, consumer and directory repository. The producer and consumer register themselves to the directory repository. The basic unit of monitoring data in GMA is called an event. It supports a request/response model. GMA comprises of three components: Producer, Consumer and Directory services.

The component that makes the event data available is called a producer, and a component that requests or accepts event data is called a consumer. A directory service is used to publish what event data is available and which producer to contact to request it. The special feature of GMA is that data transfer directly from producer to consumer[10].

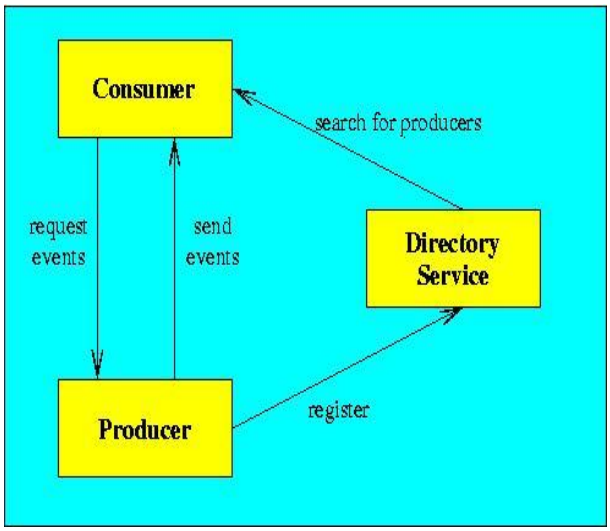


Figure [1]- Grid Monitoring Architecture

3. GriFT

GriFT (Grid Fault Tolerance) is a client- server application designed to monitor small networks from server end. It is designed using Java programming language which makes it a multiplatform monitoring technique. The technique has been derived from the existing monitoring techniques Nagios and Ganglia. The implementation includes the monitoring of all the nodes in the network and analysing their detailed performance for fault tolerance. GriFT technique monitors each and every connected node and pushes the detailed information of it to the Server System. Based on this information we could easily find the node where the fault exists all the nodes in the network and analysing their detailed performance for fault tolerance. .

Comparison of GriFT with Nagios and Ganglia monitoring techniques[11]:

Services provided	Nagios	Ganglia	GriFT
Multiplatform	Via-plugin	No	Via plugin
IP SLA	Via- plugin	No	Via plugin
Trending	Yes	Yes	Yes
Logical Grouping	Yes	Yes	Yes
Auto discovery	Via- plugin	Via Gmond	Yes
Agent	Yes	Yes	Yes
Sys logs	Via- plugin	Yes	Yes
Alerts	Yes	No	Yes
Distributed monitoring	Yes	Yes	Yes
Access coverage	Yes	No	Yes
Graph	Via-plugin	Yes	Via-plugin
Language	C, perl	Perl, C	Java
Simulator	Supports	Supports	Not required
Architecture	Client-server	Peer-to peer	Client-server
Applicable to	Grid computing, cloud computing	Grid computing, cloud computing	Small Networks

Table 1- comparison of GriFT with Nagios and Ganglia

4. PROPOSED METHODOLOGY

Following is the proposed methodology that we adopted to develop the GriFT monitoring technique:

1. Arranged a laboratory setup in the campus that creates a dynamic environment.
2. Find a monitoring technique that goes well to monitor faults in the environment. For this, we proposed a new technique for the setup.
3. Next we developed the technique using Java language and corresponding API's.
4. Technique is implemented on the setup.
5. The results are analysed whether it goes well with our requirements.

5. IMPLEMENTATION

Grid Fault Tolerant (GriFT) is a system designed to monitor the small network systems from the Server End. The architecture constitutes of two different parts and is based on the Client to initialize the communication:

- Server End Application (GriFT Server)
- Client End Application (GriFT)

GriFT Server- GriFT server implements the server architecture and is installed on server machine. It fetches the complete detailed information from all the working client nodes involved and can be viewed graphically. Working of three parameters has been monitored here- CPU Usage, Memory usage and Processes running. A threshold value (T) is assigned to every parameter (CPU usage, Memory usage and Processes) [12]. If the performance (P) is below the threshold value it will show OK state(green colour in pie chart) , if it goes above soft threshold (Ts) value then it will show the Warning state (yellow colour in pie chart) and if goes above hard threshold (Tmax) value, then a critical situation (orange colour in pie chart) will arise. i.e

Warning/Soft threshold value (Ts) = 85 (Processes)
Warning/Soft threshold value (Ts) = 60 (CPU and Memory)
Critical/Hard threshold value (Tmax) = 80 (CPU and Memory)
Critical/Hard threshold value (Tmax) = 95 (Processes)

P > Ts = Warning
P > Tmax = Critical
P < Ts = Ok

Different classes have been maintained to push information from the clients, to show particular instance of the node or to save the log files generated, to show defective nodes present on the network and to present graphical view. GriFTEngine is our main class where in a JFrame GriFT Server is displayed with drop-down JMenus:Monitor Panel: it is used to either start or stop the server.

Figure [2]- Start-Stop Frame

GriFT system in order to monitor over the network systems gets the complete information of :

System Info- This module of the GriFT System give the detailed information of the Hardware constituting the client node.

Figure [3]- System info View

CPU Info: This module of the GriFT System gives the detailed information of the CPU of the node along with the total idle time, I/O wait time, service request time, etc.

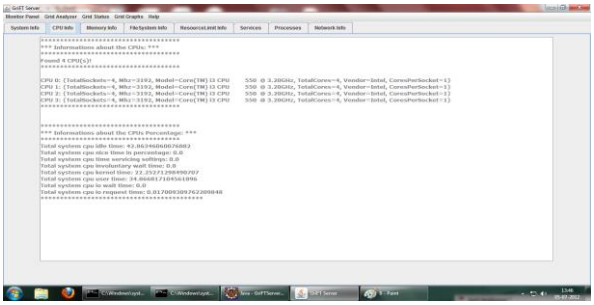


Figure [4]- CPU info View

Memory Info View- This module of the GriFT System gives the detailed information of the Memory available with total free memory, used memory, Random Access memory.

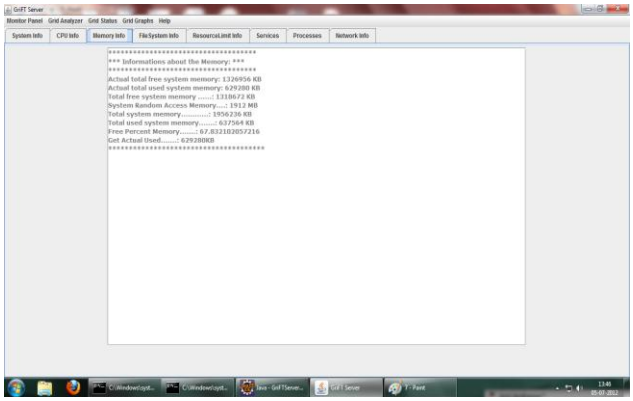


Figure [5]- Memory Info view

File System Info: File System Info module gives the detailed information about the available File System and the no. of drives available.

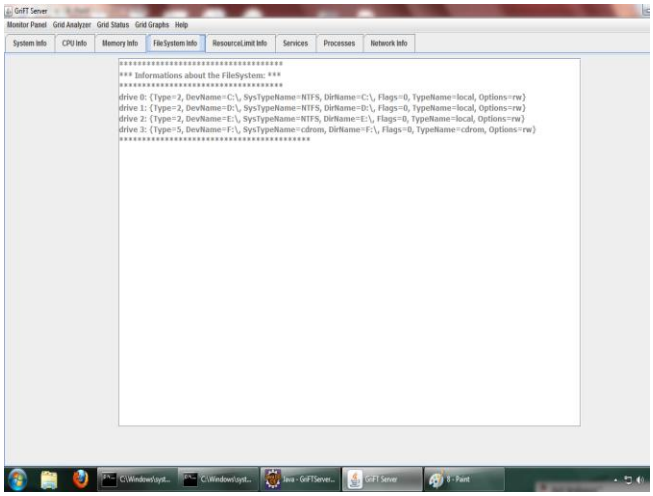


Figure [6] -File system info

Resource Limit Info: Resource Limit Info gives the complete information of the available Resources

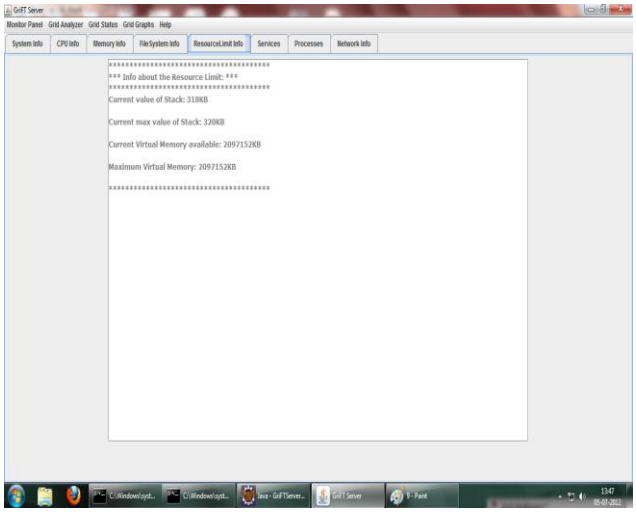


Figure [7]- Resource limit info view

Services Info: Services Info part of the module gives the detailed information about the services along with the state of the service at the client node

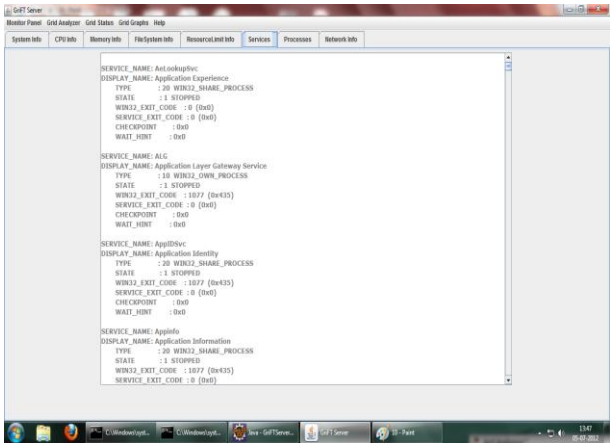


Figure [8]- Services Detail View

Process Info- Process Info gives the detailed view of the running processes at an instance and memory used.

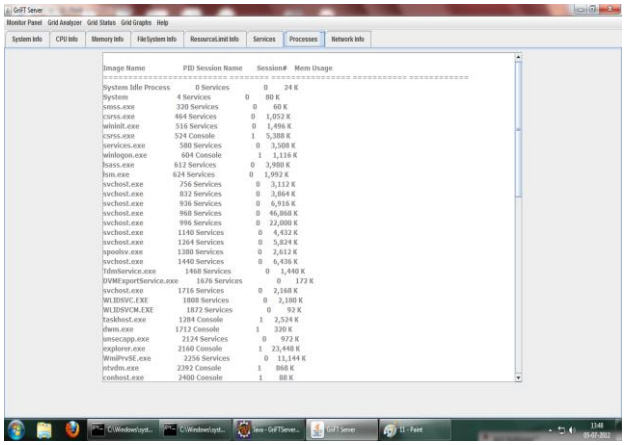


Figure [9] Processes running

Network Info: Network Info gives the detailed view of the active connection along with the protocols being used. It also gives the port no. on which the connections are made.

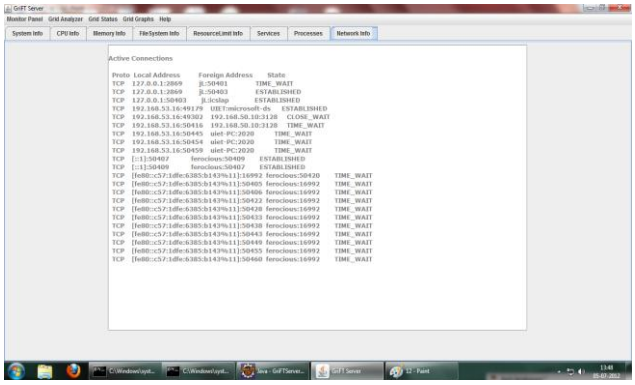


Figure [10] Network info View

With this specific information we can see the detailed information of the particular client.

Grid Analyzer: this drop-down displays the detail view of all the nodes and on further clicking the particular node it

represents the whole information about the node like system information, resource limit information, processes, network information, filesystem information and CPU information. User can also explode history of the respective node where the log files are displayed from the time it is started.

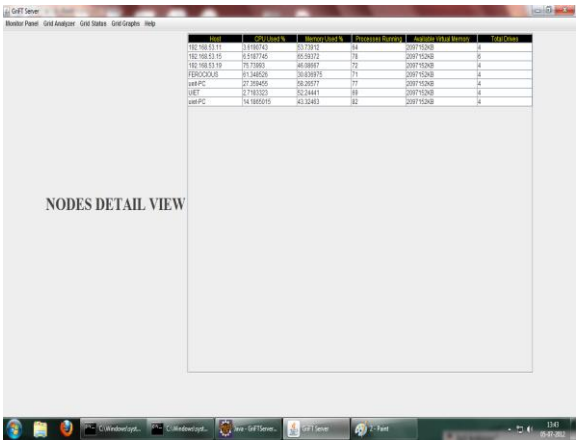


Figure [11]- Grid Analyzer

Grid Status: this JMenu displays the latest status along with the last time checked.

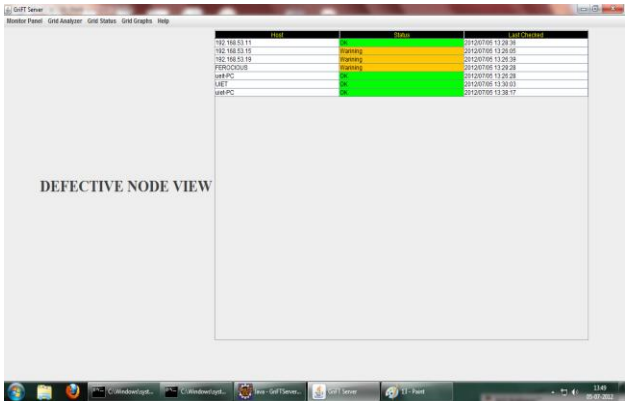


Figure [12]- Grid Status

Grid Graphs: the Grid Graphs gives the graphical representation of the overall network in the form of Pie-chart along with its status. And a line chart view is used to show the status of the specific node for its Memory and CPU usage at x-axis and time at y-axis.

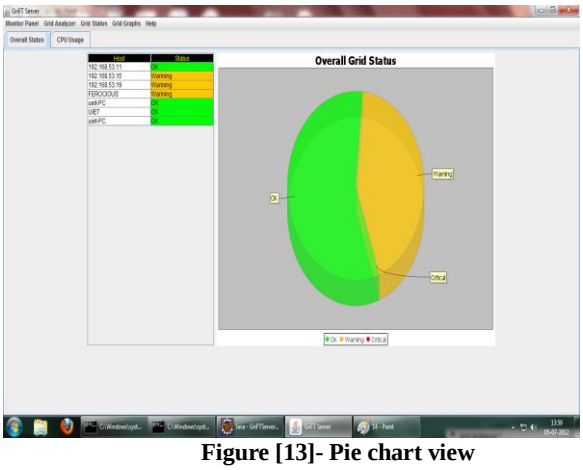


Figure [13]- Pie chart view

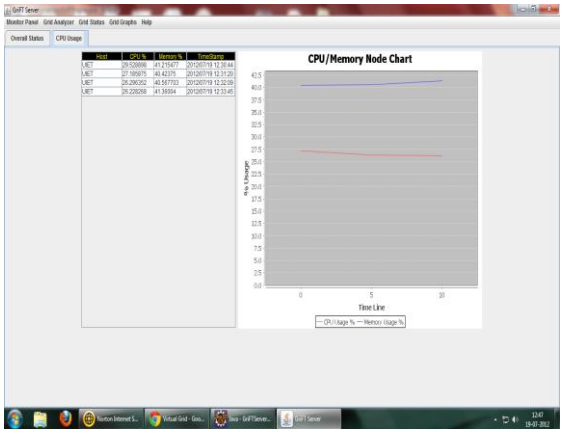


Figure [14]- Line chart View

GriFT Client- GriFT client is a client side application which is required to be installed on every client node to be monitored. It pushes the detailed configurational information from client and passes it to the server by heart-beating after particular time span. Heart-beating is the process of continuously sending heart-beats/ signals to the demanding destination. Each heart-beat contains a time stamp representing the start of the instance.

Various classes have been designed for client application to fetch information regarding operating system, network status, processes and services running and system profiler.

Results

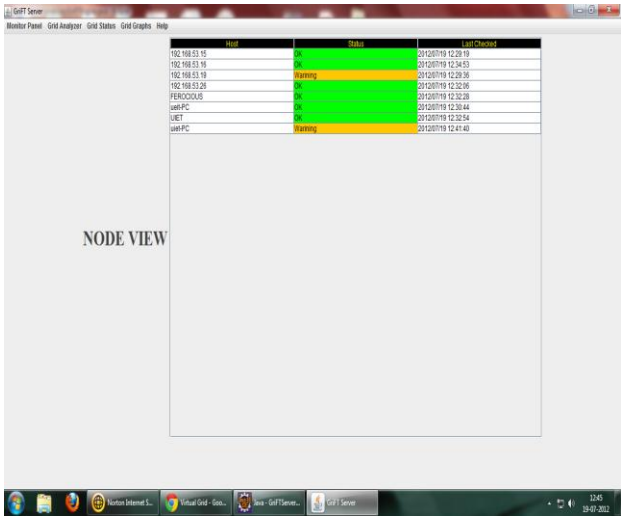
In order for a monitoring technique to be widely used, it must be performed on a local area or splitting the system into small units, so that it can scale well. The performance analysis of the GriFt monitoring technique is done at the laboratory setup arranged in our department over a network of Eight nodes (it can be extended to 100 as defined in the system by the concept of serialization in which we can create batches of seven-seven nodes and send their report to the corresponding server, we could analyze their performance by making them to push over the network to the common centralized authority).

The computers on the network were almost of the same configuration but are installed with different software's and while checking their performance different applications are made to run on client machines.

The performance of every node is analysed, it can be extended to 100 nodes as defined in our system. However in order to increase the number of nodes we can change the defined threshold value of the number of systems connected in ServerSocket().

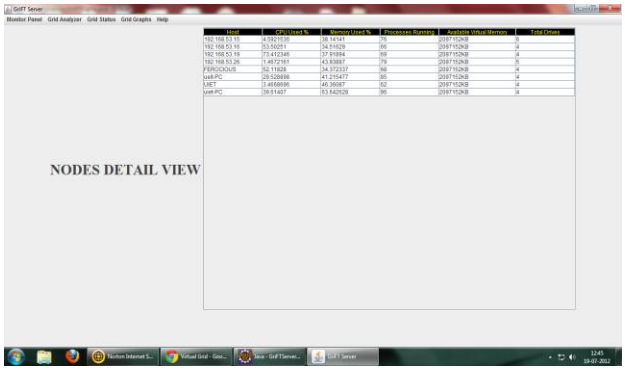
In order to cater the issue of congestion we can change the value of ServerSocket .setsoTimeout(). The timeout values can be defined similar for a batch of nodes, so the information pushed at a particular time frame for the number of nodes will be equivalent to the number of nodes in that particular batch of the client system.

The performance is analysed after installing server application over server machine and client application over the connected nodes. The snap shots were taken after 5 seconds of time stamp and the following status was found:



Fig[15]- Node view

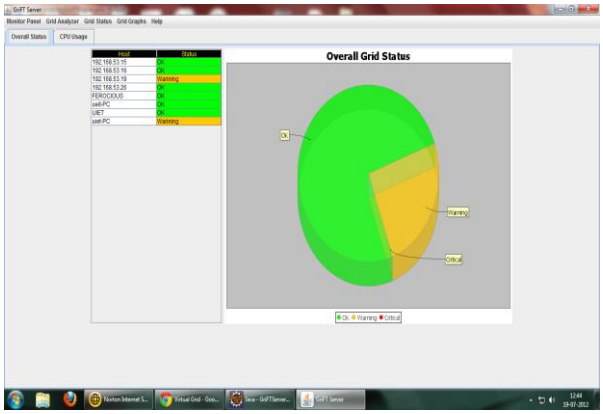
In this figure we can see two nodes are showing warning state. We can check the location of their fault by clicking at the particular node



Figure[16]- Nodes Detail View

With this figure we have analysed that- in node 192.168.53.19 the CPU usage is going above our threshold value and remaining fields are working below the threshold value. And in node uiet-pc memory usage is above the threshold value and the processes running are also at critical level. With this information we analysed that both the nodes require attention at specified areas.

The Overall Network status can be viewed in Pie-chart



Figure[17]- Overall grid Status

Now, we know the individual status of each node. We could visit each node to get the detailed view and the history of each node. Thus we will click on CPU usage in Pie-chart view which will give us the status of the CPU percentage usage and Memory percentage usage at the last time stamp instance when the dataset was heartbeat by the individual node to the server system. Since we may like to know the history of the node and how over the time the node has responded to the load. Thus, we click on the particular IP address or name of the node of our concern.

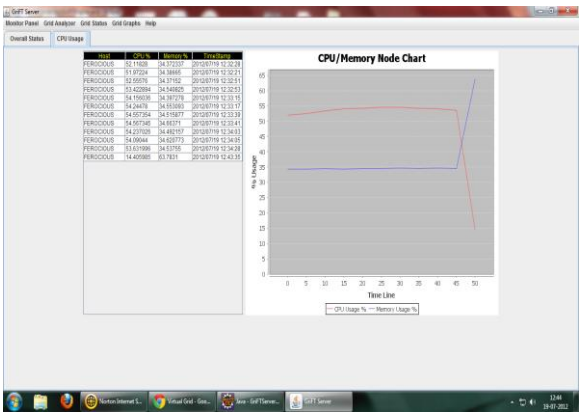


Figure [18]- Line chart view of Node Ferocious

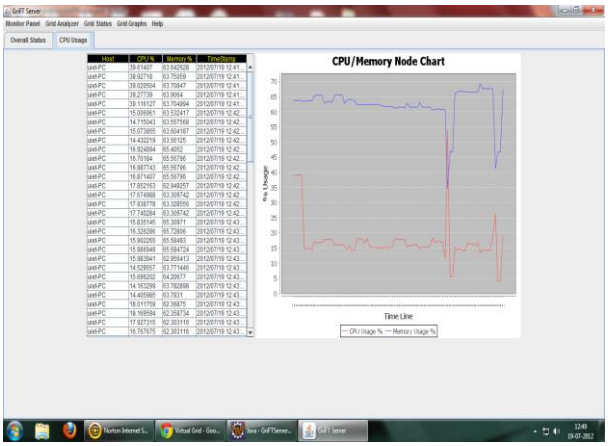


Figure [19]- Line chart view of node uiet-PC

In this way we analysed all the eight nodes and compared their performance graphs with the task manger, and we got the same results of both the applications.

6. CONCLUSION

Based upon our study of fault tolerance and monitoring techniques in grid computing, we developed a technique called GriFT(Grid Fault tolerance) for monitoring small networks. We used Hyperic Sigar API in order to monitor data which includes system information, network information, and much more. It is used to perform hardware level programming as Java does not have flexibility to go directly to hardware level programming. Since, we have used Java programming language which makes our application a platform independent application. GriFT inspite of the platform on which it run fetches information from all the connected nodes. GriFT technique monitors each and every connected node and pushes the detailed information of it to

the Server System. Based on this information we could easily find the node where the fault exists.

The results present the graphical representation of the status of all the nodes at particular instance of time and store the information in system log files giving the detailed information about every node involved. The information retrieved here is CPU usage, Memory usage and the number of processes running. A threshold value is assigned to each parameter beyond which it will warn the administrator about its condition. And it is up to the administrator to rectify the situation.

It can be concluded as the monitoring technique used by the administrator for the performance analysis of a network. Administrator observing can check the overall performance of the network and also can observe the status of the specific node. The GriFT monitoring technique can be used:

- During the time of online examination for performance analysis of the systems in the campus where compromising with faults is of no chance.
- In Network System wherein the performance of the overall system matters and it does not matter wherein the service request was fulfilled.
- For designing Self Healing Systems where in after finding the defective node from GriFT the system could go in for Self Healing.
- Asymmetric Load Sharing, hence the unutilized systems could be given high ratio of work rather than to the optimum and over utilized systems, increasing the level of performance.
- For load balancing to divide data flow to different network nodes so increase the service time.
- Continuous monitoring can help us find the nodes which have stopped responding and hence we may stop sending the load or network traffic, and once it becomes online we may start sending the load or traffic to the node.
- Storing the information for future use in problem diagnosis.

Future Scope

The research work can be extended in many directions:

- The current version only works over the LAN ; we can extend it to a wide area for systems outside the Local Area network, and at remote Locations or to a Grid.
- In this we have only studied about monitoring, we can take it to next level by adding functionality of recovering from the failure.
- Our implementation is manual. There must be some provision that whenever we client login it will automatically start working.
- Addition can be done in kind of alert; it can alert the client by e-mail or sms.
- In order to service the remote clients rather than going personally to the system we could service the system using the help of Remote system sharing or using tools like VNC Server, etc.

7. REFERENCE

1.) Martin, N. H. (September, 24, 2004). "*Grid Computing: Harnessing Underutilized Resources"*. UNCW Department of Chemistry & Biochemistry Seminar.

2.) R. Buyya, S. Venugopal. (2005). *"A Gentle Introduction to Grid Computing and Technologies"*. CSI Communications , 9-19.

3.) J.C Durand (November 8,2004). *"Grid Computing: A Conceptual and Practical Study"*, University of Lusanne.

4.) Inderpreet Chopra. *"Fault tolerance in computational Grids"*, MS Thesis Thappar University.

5.) Paul Townend and Jei Xu(2003). *"Fault Tolerance within a Grid Environment"*, *Proceedings of AHM 2003.*

6.) Yuuki Horita, Kenjiro Taura, Takashi Chikayama(2005). *"A Scalable and Efficient Self-Organizing Failure Detector for Grid Applications"*, 6th IEEE/ACM International Workshop on Grid Computing.

7.) Retrieved July 2012, from http:// xml.csie.ntnu.edu.tw/course/ftol/10172001.doc

8.) Dan Gunter, Brian L. Tierney, Craig E. Tull, Vibha Virmani(June 16, 2003) *"On-Demand Grid Application Tuning and Debugging with the NetLogger Activation Service*".

9.) Serafeim Zanikolas, Rizos Sakellariou(July 21, 2005). *"A taxonomy of grid monitoring systems"*, Future Generation Computer Systems pp163–188.

10.) B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski(March 2000). *"A Grid Monitoring Architecture"*, Global Grid Form.

11.) Wikipedia.org(July 2012). "*Comparison of Network Monitoring systems* ".

12.) M. Bubak, T. Szepieniec, M. Radecki(2003). **"***A Proposal of Application Failure Detection and Recovery in the Grid"*, Cracow Grid Workshop.

13.) Matthew L. Messie, Brent N. Chun, David E. Culler(June 2004). *"The Ganglia Distributed Monitoring System: Design, Implementation and Experience"*. Elsevier Open Access, www.Science Direct.com.

14.) Amit Jain and R.K. Shyamasundar(2004). *"Failure Detection and Membership Management in Grid Environments"*, Fifth IEEE/ACM International Workshop on Grid Computing pp 44-52.

15.) CDAC Experts (August, 2004). "*An overview of grid computing workshop"*.

16.) Manjula K A, Karthikeyan P (2010). *"Grid Computing-A tool for enhancing the computing power"*, Indian Journal of Computer Science and Engineering.