A Partial Self Healing Tool in Autonomic Computing

N. Rukma Rekha and M. Surendra Prasad Babu

***Abstract*—Self Management is a process by which computer systems shall manage their own operation with out human intervention. Self Management technologies are expected to pervade the next generation of computing or communication systems. Currently, the most vital industrial initiative towards realizing self management is the autonomic computing (AC). Its ultimate aim is to innovate the incredible self management systems capable of high level functioning in handling tremendous complexities of the management while hiding the system's complexity from the user. Self Healing, an emerging research discipline is considered as one of the key autonomic computing attribute. This paper emphasizes on developing a self healing tool operating in client server mode. The health monitor (HM) embedded in each client is the focal element of the Self Healing and plays a dominant role in healing the processes by maintaining the status of the processes periodically. If the HM fails, then the main monitor (MM) projects the task on healing the HM. And finally the external monitor (EM) adds a promising value for database consistency. In this paper, we implemented a prototype of Health Monitor as a self healing tool component. Thus all these three modules have to concurrently cooperate with each other as part of self managing.**

***Index Terms*—Self healing; autonomic computing; self management; health monitor**

I. Introduction

Systems managing themselves as per administrator's goals and integration of new components into a network seamlessly are the visions of self-managing computing systems. The main obstacle for further progress of IT industry is a looming software complexity crisis [1]. The applications and environments that weigh in at tens of millions of lines of code and require skilled IT professionals to install, configure, tune, and maintain. The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Computing systems complexity appears to be approaching the levels of human capability, yet the march toward increased inter-connectivity and integration rushes ahead undebated. Relying alone on Programming solutions will not help us from getting out of this crisis. The complexity crisis can be tackled by designing the self managing machine embedded with automated functions. Autonomic computing is an evolution to cope with rapidly growing complexity of integrating, managing and operating computing based systems. The realization of autonomic computing will result in a significant improvement in system management efficiency. An autonomic computing system is context aware and responds correspondingly in its environment. The disparate technologies that manage the environment work together to deliver best performance results.

A. Motivation:

Computer systems are becoming more demanding, complex, challenging and difficult to manage. The data to be processed is moving from more to huge. The world of Internet has given a programmer to do more complex and demanding software solutions. As networks and distributed systems grow and change, system deployment failures, hardware and software issues, and human error can increasingly hamper effective system administration. As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at run time. The number of systems will become too massive and complex for even the most skilled system integrators to install, configure, optimize, maintain, and merge and there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands. The solution to overcome for handling such difficult complexities is the Autonomic computing [2]. Autonomic computing (AC) helps to address the complexity issues by using technology to manage technology. Self healing, an autonomic attribute is dealt in this paper.

B. Objective:

The pivotal objective of the work focuses on developing an innovative self healing tool operating in a client server mode. Its main objective is to add sophisticated functions to the Health Monitor that results in a powerful self healing tool. The research also targets for a system to accomplish the vital self healing attribute of the autonomic computing.

II. Back Ground

A. Autonomic Computing (AC):

Automating the management of computing resources is not a new problem for computer scientists. For decades they have been evolving to deal with the increased complexity of system control, resource sharing, and operational management. Autonomic computing is just the next logical evolution of these past trends to address the increasingly complex and distributed computing environments of today. The radical changes in the information technology in the few short years since the mid-1990s, dramatically led to larger scale, broader reach, and a more mission-critical fundamental requirement for e- business. In that time the norm for a large

Manuscript received September 10, 2011; revised December 16, 2011.

N.Rukma Rekha is with Department of CIS, University of Hyderabad, Hyderabad, India (e-mail: rrcs@uohyd.ernet.in).

M.Surendra Prasad Babu is with Department of CSSE, Andhra University, Visakhapatnam, India (e-mail: drmsprasadbabu@yahoo.co.in).

on-line system has escalated from applications such as networks consisting of tens of thousands of fixed-function automated teller machines connected over private networks to rich suites of financial services applications that can be accessed via a wide range of devices (personal computer, notebook, handheld device, smart phone, smart card, etc.) by millions of people worldwide over the internet. We have basically five defined [2] evolutionary levels stating - level 1 as the basic level that presents the current situation where the machines are managed manually. Level 2-4 is a partial autonomic machine embedded with some automated management functions. Level 5 represents the ultimate goal of autonomic computing as the self managing machine. Self management is a process by which computer systems shall manage their own operation with out human intervention. Like the biological namesakes, autonomic systems will maintain and adjust their operation in the face of changing components, workloads, demands, and external conditions and in the face of hardware or software failures, both innocent and malicious. Self-managing systems and devices will seem completely natural and unremarkable, as will automated software and middle ware upgrades. Autonomic computing systems would control the functioning of computer applications and systems with out input from the user. Many of the decisions made by autonomic elements in the body are involuntary, whereas autonomic elements in computer systems make decisions based on tasks chosen to delegate the technology [3]. The fundamental attributes are self-configuring, self healing, self-optimizing, and self-protecting [3] [4].

All these attributes are constituted to form a self managing system or an autonomic machine. Other attributes are self aware, environment aware, self monitoring, self adjusting etc. Autonomic computing helps to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth. An autonomic system makes decisions on its own, depending on its policies, it will constantly check and optimize its status and automatically adapt itself to changing conditions [5].

B. Need of Autonomic Computing:

The growing complexity of modern networked computer systems is currently the biggest limiting factor in the expansion of corporate world. The increasing heterogeneity of big corporate computer systems, the inclusion of mobile computing devices and the combination of different networking technologies like WLAN, cellular phone networks, and mobile networks make the conventional, manual management very difficult, time consuming, and error-prone. Simply stated from above, managing complex systems has grown too costly and prone to error. It is now estimated that one-third to one-half of a company's total IT budget is spent preventing or recovering from crashes [6]. Aberdeen group studies show that administrative cost can account for 60 to 75 percent of the overall cost of database ownership (this includes administrative tools, installation, upgrade and deployment, training, administration salaries, and service and support from database suppliers).The autonomic computing reduces deployment and maintenance cost and increases stability of IT systems that are able to adopt and implement directives based on business policy, and are able to make modifications based on changing environment.

The goal of the Autonomic computing is to create systems that run themselves, capable of high-level functioning while keeping the system's complexity invisible to the user. Its challenge is to produce practical methodologies and techniques for development of such self managing systems, so that they may be leveraged to deal with failure and recover easily.

C. Challenges of Autonomic Computing:

The Autonomic computing must result in smooth management of business by minimizing the complexities of the users. The other challenge is to convince customers that Autonomic computing actually simplifies systems management and can cut costs to a large extent. The transition of new self-healing systems must cause minimal or no disruption. The vision of Autonomic computing [7] is to improve performance of IT systems by introducing self management systems for configuration, protection, optimization and healing purposes.

III. Literature Survey

Goal of self managing systems is to introduce autonomic computing into personal computing platforms, given that users will continue to manage their own platforms, and need to know what state they are in and how they got that way.

A. Personal Autonomic Computing (PAC):

Personal autonomic computing [8] is autonomic computing in a personal computing environment. Personal computing is still too demanding of its users, especially when it comes to setting up, managing and fixing the software and data that are so vital to its value. Although improvements have been made, the cost of personal computing is still dominated by the time its end users must spend maintaining and configuring its software. Autonomic computing with its emphasis on automation of system management has the potential to reduce this time, simplifying personal computing and making it more robust and secure.

Most early efforts in Autonomic computing have been focused on servers. For some customers, servers have become almost unmanageable because of sheer complexity coupled with business-critical and escalating demand. In some respects, achieving autonomic computing with in server environments will be an easier task than with in personal computing. Servers are likely to have received the level of investment to ensure in-built fault tolerance and include extensive redundancy. Personal devices are often machines built on faster, cheaper and smaller philosophy with limited resources. Other considerations are required for personal computing such as flexibility of location (e.g. laptops) and of hardware (e.g. palm devices) and software configuration that complicate further goal of achieving autonomic computing [8].

The challenge of autonomic personal computing is to simplify and enhance the end user experience, delighting the user by anticipating his or her needs in the face of this

complex, dynamic and uncertain environment.

B. Self Healing:

Major portions of corporate worlds time is wasted in identifying, tracing, and determining the root cause of failures in complex computing systems. Serious customer problems can take teams of programmers several weeks to diagnose and fix, and sometimes the problem disappears mysteriously with out any satisfactory diagnosis [6].

The sharp difference between the current computing and autonomic computing towards self healing concept is that the current computing involves problem determination in large, complex systems can take a team of programmers weeks where as in the autonomic computing involves system automatic detection, diagnosis, and repairs for localized software and hardware problems. Systems will be predicting the problems in advance and will take actions to prevent the failure from having an impact on applications. The Self-healing objective is to reduce all possible disruptions in order to keep enterprise applications up and available at all times.

Developers of system components need to focus on maximizing the reliability of each hardware and software product toward continuous availability. Systems will discover faults, diagnose them, and react to disruptions. For a system to be self healing, it must be able to recover from a failed component by first detecting and isolating the failed component, taking it off the network, fixing or isolating the failed component, and reintroducing the fixed or replacement component into service without any apparent application disruption. Systems need to predict problems and take actions to prevent the failure from having an impact on applications. The self-healing objective must be to minimize all outages in order to keep enterprise applications up and available at all times. Developers of system components need to focus on maximizing the reliability and availability of each hardware and software product toward continuous availability.

C. Past Work:

[9] Develops a proof of concept of self-healing tool for personal computing environment operating in a peer-to-peer mode and consists of a pulse monitor and a health monitor. This provides the feasibility of the pulse and vital signs concepts and their ability to provide some self-healing properties within a personal computing environment.

C.1 Peer-To-Peer (P2P):

Peer-to-Peer (P2P) is a paradigm in which each workstation on a network has equivalent capabilities and responsibilities [10]. This differs from the Client/Server architecture, in which a server is a dedicated computer to serve client requests. Peer to peer computing offers a company a cost-efficient way of sharing computer resources, improving network performance, and increasing overall productivity [11].

In traditional peer to peer networking, computers are connected together as a workgroup and configured for the sharing of resources such as files and printers. Peer to peer architecture enable computers to share services and resources directly between one another. Peer to peer technologies benefit distributed computing as it provides efficient communication and quality of service. Here, the peers form a 'neighbour-hood watch' scheme-- looking out for each others health [12]. A system has its autonomy to register and un-register with other systems. Two systems become neighbours after they register to each other. There is no limit on how many neighbour(s) that a system can register with. Systems send pulses to each other only when they are connected. Therefore, a system does not necessarily always have to be available, as a registered system it may find another peer. Peer to peer computing also allows networks to work together using intelligent agents [11].

C.2 Self Healing Tool Design In Peer to Peer:

The assumption behind the tool is that dying/hanging processes on the PC have indicators to the health of that PC. Pulse monitor and health Monitor are the fundamental modules of this design. The vital signs may indicate that the PC is becoming unstable and possibly in immanent danger of hanging or unreliable for current processes running on that machine. As well as restarting the detected hung process (es) the peers are notified of the situation via a change in pulse. This is particularly useful in situations where the PC is unattended for example running on a web server, and the user may be notified via a peer PC that the machine is in difficulty. Another user situation is when machines in the peer group are sharing workload, for example via harmony PC grid services; a peer is notified in advance of immanent danger and can recover data and re-allocate work to another peer. Such an approach is more proactive than responding once the machine has hung, and as such offers fuller potential for autonomic capabilities. The underlying functionality of the tool is a heartbeat monitor; if a process hangs it should be restarted and the pulse monitor takes note. Upon several processes hanging with in specified time frames, a change occurs in the monitor's perception of how healthy the machine is and as such brings about a change in the pulse being broadcast from that PC. Since this tool operates in a peer to peer mode, it also takes responsibility to watch out for its neighbours, as such others PCs (peers) will register with it and it will monitor their pulse. An Internal Monitor inside takes care of pulse. Each is able to send its pulse to a peer via an external monitor.

The internal monitor sends the degree of urgency (nominal, interesting, important, urgent, no pulse) to the peer's pulse external monitor instead of just a 'beat'. The urgency level is transformed on the number of failed processes. The amount of processes required to cause a change in pulse is adaptable and need not necessarily remain at the values as is the time window for qualifying processes. In short, the tool scans the system periodically to check its health condition; it transforms the health condition to a pulse value and will send it to connecting neighbours. If a process is found to have failed, the tool will try to re-start that process.

Pulse monitoring self healing tool contains four components: Main monitor (MM), internal monitor (IM), external monitor (EM), and health monitor(HM)[9]. IM and HM monitor processes on a machine. HM can re-start or terminate a failed process. EM communicates with the external environment, it sends/receives Pulses to/from other systems (neighbours /peers); monitoring neighbours by

sending a message to check if the neighbour is 'alive' or not when it detects that the neighbour hasn't sent it pulse and reboot a neighbour when necessary. Conversely, the system is being monitored by its neighbours in the same way. MM is responsible for monitoring IM and EM. MM would re-start them if they are dead.

D. Origin of the Current Research:

The concept of the current research in built is based on the above paper [9]. For user-friendliness, same names were given to the modules in the current research also. How ever, the functionalities of the some modules vary in the current research. In addition to this, there are some bottom-line differences between these two researches. The current research focus on designing a self-healing tool in client server mode where as the past research is designed in peer-to-peer mode. The current research considers only three modules namely health monitor (HM), main monitor (MM) and external monitor (EM).

IV. Architectural Design

Client-server computing or networking is a distributed application architecture that partitions tasks or work loads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware.

A server is a high-performance registering unit and shares its resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests [13].

Client-server describes the relationship between two computer programs in which one program, the client program, makes a service request to another, the server program. Each instance of the client software can send data requests to one or more connected *server*s. In turn, the servers can accept these requests, process them, and return the requested information to the client. The two tier architecture means that the client acts as one tier and application in combination with server acts as another tier.

The three modules of this design perform specific tasks as

1) Health monitor (HM)
2) Main monitor (MM)
3) External monitor (EM)

The following figure depicts the design architecture briefly.

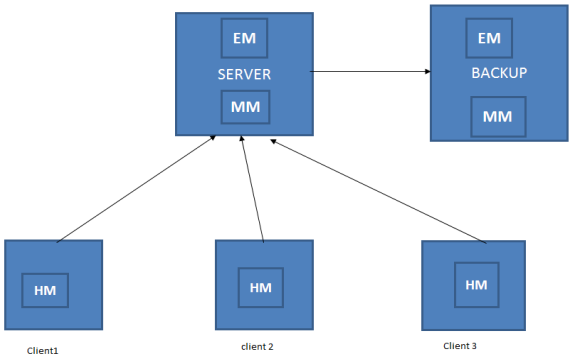

Fig: Overview of the architecture

A. Strategy:

Consider three systems represented as client c1, client c2, and client c3 as shown in figure. The health monitor is embedded in each client. Every HM is confined to its own client. The HM of one client is not related to any other client although the function of the HM is same in all clients. Initially the users can perform their own tasks on these systems. Let there exists some user processes running on these systems. These user processes after some period of time, by chance may stop working when a unknown problem arises at some point. This may lead to the processes failure or termination that completely kills the precious time and waste the resources. Thus the HM of these clients takes the responsibility in handling such faulty process and possibly does a necessary healing action based on the problem encountered by the process. That's the reason for the HM to be the pivotal module in healing the processes.

There may be some circumstances that may result in the failure of the HM. At this point of time the main monitor (MM) which is introduced in the server takes the responsibility of tackling the HM. A question may arise at this moment regarding how the MM knows which HM is in failure state. As this research is based on client server technology, the HM will send a pulse to the server periodically. Here the pulse is “I am alive” message. This message is sent to the MM at regular intervals. If this message is not received by the MM, then the MM simply recognizes that a specific HM is in failure state and immediately takes a action in restarting the HM.

Thus if there is any fault in the process the HM will handle it. And if this HM fails then the MM in the server will tackle that. And when the serves crashes the backup server will look after that. Thus when ever the server crashes the backup server comes into picture and ensures the adaptability to the clients to continue their tasks with out interruption. And this backup server acts as the original server right now.

Finally the external monitor (EM) which is embedded in both the servers performs a peculiar task. Its job is to maintain thedatabase consistency in both the servers so that the clients can access the updated information at any point of time even though one of the servers is crashed.

B. Functionalities of the Modules:

B.1 HEALTH MONITOR (HM):

The self healing tool health monitor (HM) performs fabulous work when compared to the other modules main monitor (MM) and the external monitor (EM). The health monitor regularly monitors the health of their concerned systems. This mainly focuses on healing the effected processes. In addition to healing, the health monitor opens new opportunities in the utilization of the resources effectively and optimistically.

B.2 MAIN MONITOR (MM):

The main monitor sole responsibility is to handle the health monitor when it fails. The MM receives the pulse in the form of message from the clients. When this pulse is not received by the main monitor, it simply restarts the corrupted health monitor.

B.3 EXTERNAL MONITOR (EM):

The task of the External Monitor is to maintain the

consistency of the database in both the servers. An immediate action should be taken at this point of time to make the server come to the normal mode. The backup server will come into role when the main or original server crashes. When the communication is switched on from main server to the backup server the clients will be able to access the updated information.

The external monitor provides the flexibility to the clients to access the updated information at any time. Log files records updated information. This updated data is permanently stored in the data files when the check point occurs. Thus whenever the check point occurs the external monitor will simultaneously reflect the log files to both the data files of the two servers and hence results in successfully maintaining database consistency.

C. Description of the HM:

Process is a running instance of a program. Every process is provided with resources to do the task. *RLIMIT_CPU* is the maximum CPU time in seconds. When the process reaches the soft limit, it is sent a SIGXCPU signal. The default action for this signal is to terminate the process. However, the signal can be caught, and the handler can return control to the main program. If the process continues to consume CPU time, it will be sent SIGXCPU once per second until the hard limit is reached, at which time it is sent SIGKILL. Portable applications that need to catch this signal should perform an orderly termination upon first receipt of SIGXCPU. *RLIMIT_AS* denotes maximum size of memory a process can take. In short it is the maximum size of the process virtual memory (address space) in bytes. This limit affects calls to brk(2), mmap(2) and mremap(2), which fail with the error ENOMEM upon exceeding this limit. Also automatic stack expansion will fail (and generate a SIGSEGV that kills the process if no alternate stack has been made available viasigaltstack(2)). *RLIMIT_FSIZE* shows maximum size of files that the process may create. Attempts to extend a file beyond this limit result in delivery of a SIGXFSZ signal. By default, this signal terminates a process, but a process can catch this signal instead, in which case the relevant system call fails with the error EFBIG. *RLIMIT_CORE* is the maximum size of *core* file. When RLIMIT CORE is zero no core dump files are created. When greater than zero, larger dumps are truncated to this size. *RLIMIT_STACK* shows the maximum size of the process stack, in bytes. Upon reaching this limit, a SIGSEGV signal is generated. To handle this signal, a process must employ an alternate signal stack (sigaltstack(2)). *RLIMIT_DATA* is the maximum size of the process's data segment (initialized data, uninitialized data, and heap). This limit affects calls to brk(2) and sbrk(2), which fail with the error ENOMEM upon encountering the soft limit of this resource.

RLIMIT_RSS: It is the maximum resident set size. It specifies the limit (in pages) of the process's resident set (the number of virtual pages resident in RAM).

RLIMIT_NOFILE: It is the maximum number of open files. Simply it specifies a value one greater than the maximum file descriptor number that can be opened by this process. Attempts (open(2), pipe(2), dup(2), etc.) to exceed this limit yield the error EMFILE.

RLIMIT_NPROC: The maximum number of processes that can be created for the real user ID of the calling process. Upon encountering this limit, fork(2) fails with the error EAGAIN.

RLIMIT_MEMLOCK: The maximum number of bytes of memory that may be locked into RAM. In effect this limit is rounded down to the nearest multiple of the system page size. This limit affects mlock(2) and mlockall(2) and the mmap(2) MAP_LOCKED operation.

The shmctl(2) SHM_LOCK locks are accounted separately from the per-process memory locks established by mlock(2), mlockall(2), and mmap(2) MAP_LOCKED; a process can lock bytes up to this limit in each of these two categories. The starting step is to collect all the user running processes and is sorted with respect to any process resource. Here the resource CPU time is taken into consideration. This phase will run until rest of the processes are executed or suspended. Let us assume there are some processes in the suspend queue. As their limits are closer to the resource limits, it clearly indicates that the processes in this queue require higher utilization of resources in order to complete their tasks. Thus the soft limits are increased closer to the hard limits. This means these processes will get privilege of using the resources massively. After increasing the soft limits the processes resume their execution. After some period of time some processes may complete their task and rest may fail or terminate. Some new processes may be created during this span of time. However they are also suspended in the new queue. In the initial phase, the reasons for the processes to fail are traced out so that in next phase the Health Monitor heals the failure processes based on the analysis done in the previous phase.

Thus after the completion of the jobs of the processes in the suspend queue, the soft limits are decreased and again the first phase will be repeated. The vital difference between the first phase and other phases is that the healing is done in all the phases except in the initial phase as analysis is done here in tracing out the reasons for the failure of the processes.

V. Implementation and Results

For user-friendly the Health Monitor strategy will be demonstrated with an example. Two queues namely suspended queue and new queue are created initially. The suspended queue contains the partially executed processes and the new queue contains the processes which are ready to be executed.

In the first step all the running processes information is collected. Let us assume that 25 processes are currently running in the system of a client. The current usage of process resources limits is periodically checked. Say, that a process reached the closer limit of the soft limit and got suspended in the suspended queue. The moment the first processes got suspended, any new process created after the suspension of the earlier process will also be suspended. But this new process will be placed in the new queue as it didn't start its job. The reason for the suspension of these processes is to make the earlier suspended processes to be executed than the later processes that are stored in the new queue. Let us consider that 5processes are closer to soft limit. As soon as

they touch the closer limit of the soft limit, they were suspended in the suspended queue. Thus the suspended queue contains 5 processes and currently there are 20 processes running on the system.

Before the execution of these remaining processes 20, say 10 new processes were created. Thus the suspended queue contains 5 processes and the new queue contains 10 processes.

After the completion of execution of these 20 processes, the soft limits are increased. They were set to very close to the hard limit. After increasing the soft limits the 5 processes that are suspended in the suspended queue are made to resume their execution making them to continue there jobs. Now these suspended processes have got the ability of higher utilization of the resources. The important thing is that when these processes resumed the execution, their status is maintained so that if any process got terminated then the process can continue its job from where it stopped as the process status is stored here time to time. This is what healing means generally.

Let us assume among the 5 processes in the suspended queue, 3 processes have done their jobs successfully and the other processes got terminated. There can be a number of reasons for the processes to terminate. Such as Total Time Limit exceeded, Memory unavailable errors, Arithmetic errors, Protection errors, Invalid instructions, Privileged instructions, I/O failures, Parent termination before child termination etc.

The reasons for the termination of the processes are traced out and a necessary healing functions and methods are adapted here in order to overcome these problems and making the processes to complete their jobs successfully.

The healing functions or methods are based on two types of resources. They are process resources and real resources. Process resources may be CPU, memory etc. and real resources may be a page, printer, or an input. Based on these healing functions, the Health Monitor heals a process by considering certain issues related to these resources. One healing functions is described here for the CPU usage limit. Thus the healing functions must be developed for all kinds of resources so that the healing can be done with little or no human intervention.

There is also possibility for the creation of new processes during the execution of suspended processes. Here also, the execution of newly created processes begins only after all the suspended processes in the suspended queue have completed their jobs. Let us consider that 10 new processes have created during the execution of suspended process. Thus the new queue contains totally 20 processes which have not started their execution. The process described so far can be treated as phase 1. Again the same process is repeated for the 20 processes which are in the new queue. As soon as phase 1 is completed the soft limits are decreased to their original values. Thus the system starts executing the 20processes. Again the current resource limits are checked periodically. The processes which touch the closer limit of the soft limit will be suspended in the suspended queue. Like this, the same process is repeated for the other processes also.

The processes that have not completed their jobs in the phase 1 can be completed in phase 2 assuming that the process gets its required resources in order to finish its job. Thus this process will be repeated until all the processes completed their jobs successfully.

A. Implementation Details:

One healing function is implemented which is based on CPU limit. This healing module provides flexibility to the users for controlling the processes in terms of CPU utilization. It is a program that will limit the CPU for any user running process. The healing function can reduce the % CPU of any process. When too many processes are running on the system this can be helpful in making the processes to be executed slowly which are not so important for the user. By reducing the % CPU utilization of a process, the CPU will be free and this can be used for other processes that are treated as important or having high priority.

B. Results:

Two terminals are used to operate this. One terminal shows the current usage of % CPU by executing the top command. And the other terminal executes the CPU limit program that will reduce the % CPU of any user process. The logic behind this is that the healing function sends two signals called SIGCONT and SIGSTOP to a process whose % CPU has to be altered. These two signals are sent periodically to that target process. When SIGSTOP is sent to a process, the usual behavior is to pause that process in its current state. The process will only resume execution if it is sent the SIGCONT signal. SIGSTOP and SIGCONT are used for job control in the Unix shell, among other purposes. SIGSTOP cannot be caught or ignored. In short, SIGSTOP tells a process to "hold on" and SIGCONT tells a process to pick up where you left off".

A application *mplayer* is taken for showing the results of the healing function CPU limit. The % CPU can be at any time on first terminal and it changes for every two seconds. On the other hand for reducing the % CPU the CPU limit is executed on the second terminal and thes results can be seen again in the first terminal. The terminals are made as the screen shots for showing the results of the CPU limit.

[Hostr@local ~]$ top
top - 05:05:42 up 31 min, 4 users, load average: 0.36, 0.29, 0.22
Tasks: 151 total, 2 running, 148 sleeping, 0 stopped, 1 zombie
Cpu(s): 8.1%us, 1.3%sy, 0.0%ni, 90.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 868060k used, 1196748k free, 89916k buffers
Swap: 0k total, 0k used, 0k free, 511236k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

3707 Host 20 0 28168 9084 4840 S 8.6 0.4 0:22.93 mplayer
2536 root 20 0 306m 27m 9.8m S 3.6 1.4 1:02.98 Xorg
3293 Host 20 0 120m 23m 13m S 3.0 1.2 0:10.30 gnome-terminal
3742 Host 20 0 2428 1084 840 R 0.7 0.1 0:01.44 top
3755 Host 20 0 220m 68m 45m S 0.7 3.4 0:03.82 swriter.bin
7 root 15 -5 0 0 0 S 0.3 0.0 0:00.06 ksoftirqd/1
1853 root 15 -5 0 0 0 R 0.3 0.0 0:02.21 kondemand/0

2949 Host 20 0 52060 10m 8632 S 0.3 0.5 0:05.77 gnome-settings-
2951 Host 20 0 59004 18m 11m S 0.3 0.9 0:09.47 gnome-panel
2952 Host 20 0 104m 31m 18m S 0.3 1.6 0:20.70 nautilus
TABLE 1:

[Hostr@local ~]$ top
top - 05:10:56 up 36 min, 4 users, load average: 0.49, 0.35, 0.25
Tasks: 151 total, 2 running, 148 sleeping, 0 stopped, 1 zombie
Cpu(s): 8.7%us, 1.8%sy, 0.0%ni, 89.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 879468k used, 1185340k free, 90436k buffers
Swap: 0k total, 0k used, 0k free, 518468k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
3707 Host 20 0 28168 9136 4876 S 9.3 0.4 0:50.21 mplayer
2536 root 20 0 309m 28m 9.9m S 3.6 1.4 1:14.53 Xorg
3293 Host 20 0 121m 25m 13m S 3.0 1.2 0:11.74 gnome-terminal
3755 Host 20 0 224m 74m 49m S 1.0 3.7 0:25.64 swriter.bin
2952 Host 20 0 104m 31m 18m S 0.7 1.6 0:22.41 nautilus
3742 Host 20 0 2428 1084 840 R 0.7 0.1 0:03.11 top
3 root RT -5 0 0 0 S 0.3 0.0 0:00. 53 migration/0
2053 haldaemo 20 0 7268 5008 4136 S 0.3 0.2 0:01.42 hald
2949 Host 20 0 52060 10m 8632 S 0.3 0.5 0:07.13 gnome-settings
TABLE 2:

[root@local project]# ./cpulimit -p 3707 -l 3
Process 3707 detected
TABLE 3:

[Hostr@local ~]$ top
top - 05:15:18 up 40 min, 4 users, load average: 0.17, 0.22, 0.22
Tasks: 154 total, 1 running, 149 sleeping, 3 stopped, 1 zombie
Cpu(s): 3.4%us, 3.0%sy, 0.0%ni, 93.6%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 887348k used, 1177460k free, 90844k buffers
Swap: 0k total, 0k used, 0k free, 518768k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
2536 root 20 0 309m 33m 10m S 3.0 1.7 1:20.46 Xorg
3707 Host 20 0 29924 10m 4892 T 3.0 0.5 1:08.52 mplayer
3293 Host 20 0 121m 25m 13m S 2.7 1.3 0:13.19 gnome-terminal
2987 Host 20 0 28848 3136 1880 S 0.7 0.2 0:15.95 gnome-screensav
3742 Host 20 0 2428 1084 840 R 0.7 0.1 0:04.47 top
2950 Host 20 0 26328 13m 8984 S 0.3 0.7 0:10.13 metacity
3030 Host 20 0 54120 13m 9520 S 0.3 0.6 0:06.31 gdm-user-switch
3755 Host 20 0 224m 74m 49m S 0.3 3.7 0:29.64 swriter.bin
TABLE 4:

[Hostr@local ~]$ top
top - 05:28:16 up 53 min, 4 users, load average: 0.35, 0.26, 0.20
Tasks: 162 total, 2 running, 148 sleeping, 11 stopped, 1 zombie
Cpu(s): 6.6%us, 1.4%sy, 0.0% ni, 91.7%id, 0.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 891436k used, 1173372k free, 92116k buffers
Swap: 0k total, 0k used, 0k free, 519536k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
3707 Host 20 0 28804 9732 4896 T 2.7 0.5 1:52.47 mplayer
3293 Host 20 0 121m 25m 13m S 2.3 1.3 0:17.88 gnome-terminal
2536 root 20 0 309m 34m 10m S 2.0 1.7 1:47.47 Xorg
3755 Host 20 0 225m 75m 49m S 1.0 3.8 0:54.87 swriter.bin
2987 Host 20 0 28848 3136 1880 S 0.7 0.2 0:21.59 gnome-screensav
3742 Host 20 0 2428 1088 840 R 0.7 0.1 0:08.72 top
528 root 15 -5 0 0 0 S 0.3 0.0 0:01.07 scsi_eh_3
2021 dbus 20 0 13468 1460 864 S 0.3 0.1 0:02.39 dbus-daemon
TABLE 5:

[Hostr@local ~]$ top
top - 05:18:37 up 43 min, 4 users, load average: 0.05, 0.16, 0.19
Tasks: 154 total, 1 running, 149 sleeping, 3 stopped, 1 zombie
Cpu(s): 3.3%us, 1.6%sy, 0.0%ni, 95.1%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 888024k used, 1176784k free, 91168k buffers
Swap: 0k total, 0k used, 0k free, 518828k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
3707 Host 20 0 29884 10m 4892 T 3.3 0.5 1:14.95 mplayer
2536 root 20 0 309m 33m 10m S 1.0 1.7 1:26.93 Xorg
3293 Host 20 0 121m 25m 13m S 1.0 1.3 0:15.18 gnome-terminal
2987 Host 20 0 28848 3136 1880 S 0.7 0.2 0:17.32 gnome-screensav
3742 Host 20 0 2428 1084 840 R 0.7 0.1 0:05.55 top
1853 root 15 -5 0 0 0 S 0.3 0.0 0:03.25 kondemand/0
2282 root 20 0 62352 28m 6288 S 0.3 1.4 0:02.29 setroubleshootd
2408 root 20 0 9980 1128 776 S 0.3 0.1 0:00.63 kerneloops
TABLE 6:

[root@local project]# ./cpulimit -p 3707 -l 8
Process 3707 detected
TABLE 7:

Hostr@local ~]$ top
top - 05:22:35 up 47 min, 4 users, load average: 0.37, 0.20, 0.19
Tasks: 158 total, 2 running, 149 sleeping, 6 stopped, 1 zombie
Cpu(s): 8.2%us, 2.4%sy, 0.0%ni, 89.4%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 888916k used, 1175892k free, 91556k buffers
Swap: 0k total, 0k used, 0k free, 519036k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
3707 Host 20 0 28672 9712 4896 S 8.0 0.5 1:27.08 mplayer
2536 root 20 0 310m 33m 10m S 3.3 1.7 1:33.59 Xorg
3293 Host 20 0 121m 25m 13m S 2.7 1.3 0:16.21 gnome-terminal
2987 Host 20 0 28848 3136 1880 S 1.3 0.2 0:18.83 gnome-screensav
3742 Host 20 0 2428 1088 840 R 0.7 0.1 0:06.83 top
3755 Host 20 0 224m 75m 49m S 0.7 3.7 0:40.02 swriter.bin
1853 root 15 -5 0 0 0 R 0.3 0.0 0:03.52 kondemand/0
2246 root 20 0 5624 1712 1464 S 0.3 0.1 0:00.06 wpa_supplicant
2949 Host 20 0 52060 10m 8632 S 0.3 0.5 0:08.96 gnome-settings-
2951 Host 20 0 59004 18m 11m S 0.3 0.9 0:13.60 gnome-panel
2952 Host 20 0 104m 31m 18m S 0.3 1.6 0:25.27 nautilus
3030 Host 20 0 54120 13m 9520 S 0.3 0.6 0:06.97 gdm-user-switch
TABLE 8:

[Hostr@local ~]$ top
top - 05:24:51 up 50 min, 4 users, load average: 0.31, 0.22, 0.19
Tasks: 161 total, 2 running, 149 sleeping, 9 stopped, 1 zombie
Cpu(s): 8.9%us, 2.9%sy, 0.0%ni, 88.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2064808k total, 890140k used, 1174668k free, 91796k buffers
Swap: 0k total, 0k used, 0k free, 519244k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
3707 Host 20 0 28672 9712 4896 S 8.9 0.5 1:36.89 mplayer
2536 root 20 0 309m 33m 10m S 4.3 1.7 1:38.42 Xorg
3293 Host 20 0 121m 25m 13m S 3.0 1.3 0:17.02 gnome-terminal
3755 Host 20 0 225m 75m 49m S 1.7 3.8 0:44.65 swriter.bin
2952 Host 20 0 104m 31m 18m S 0.7 1.6 0:25.86 nautilus
2987 Host 20 0 28848 3136 1880 S 0.7 0.2 0:19.79 gnome-screensav
3030 Host 20 0 54120 13m 9520 S 0.7 0.6 0:07.39 gdm-user-switch
3742 Host 20 0 2428 1088 840 R 0.7 0.1 0:07.58 top
TABLE 9:

To kill the process
[root@local project]# kill 4016
TABLE 10:

[root@local project]# ./cpulimit -p 3707 -l 3
Process 3707 detected
Process 3707 dead!
Warning: no target process found. Waiting for it...
TABLE 11:

A. *SUMMARY:*

An small application '*mplayer'* is taken into consideration for showing the working of the cpulimit. A song is played by using mplayer. The current usage of % CPU of this process can be seen in the first terminal by simply executing the top command. Table 1.shows that this process is currently using 8.6 % CPU. As this change for every two seconds, the other value of % CPU for the mplayer process is 9.3. And it is shown in Table 2. Thus the mplayer process uses % CPU in between 8and 10 approximately. If any user feels that this is not important or he would like to reduce the % CPU of this processes, then he can run the CPU limit program on the second terminal as shown in the Table 3. This is the input of the program which would like to reduce the % CPU to 3. Thus in the first terminal we can then see the output that the % CPU from 9.3 is dropped to 3. And this is shown in Table 4. Asthis changes for every two seconds the value of % CPU after two seconds is 2.7 as shown in Table 5. After two seconds the % CPU is again changed to 3.3 as shown in Table 6. This means when the SIGSTOP signal is sent to the process, the % CPU is dropped to 2.7 and when SIGCONT signal is sent to the process the % CPU is again increased to 3.3. This is how the % CPU can be controlled by an user. Again if we want to increase it then with the input as shown in Table 7, the % CPU will be raised to 8. After two seconds this % CPU have the value 8.9. This is how the % CPU can be controlled for any user process. If the application is killed with the input as shown in Table 10, the output is shown in Table 11 showing the process is dead. Like this, we can limit the percentage from 0% to 100%, which means that if you set for example 40%, your process cannot use more than 400 ms of cpu time for each second. This program can be executed with the root privilege only.

VI. Conclusion and Future Work

The growing complexity of modern networked computer systems is currently the biggest limiting factor in their expansion. Computer systems are becoming more demanding, complex, challenging and difficult to manage to cope with the new technologies. Autonomic computing helps to address the complexity issues by using technology to manage technology. Autonomic computing systems would control the functioning of computer applications and systems with out input from the user. A computer system is self-managing if it has the autonomic attributes or capabilities like self-configuration, self-healing, self-optimization, self-protection, self-aware, self-monitoring, and self-adjusting. Self-healing also called as holistic computing is concerned with ensuring effective recovery when a fault occurs with out human interaction. In a biological system, the human body reacts with the external environment involuntary; while in the computer-based systems, autonomic elements make decisions based on the available technologies. The objective of the research work was to develop a proof of concept self-healing tool for the autonomic computing environment in a client server mode consisting of health monitor, main monitor and external monitor. The health monitor is responsible for the monitoring of processes running on the machine. In short, it monitors the health of their respective clients.

As a tool the self-healing prototype could be expanded in many ways. The health monitor could be extended to detect which operating system it is running on and to call the corresponding function to obtain process information. The strength of the self healing tool depends on the efficient healing functions embedded in the health monitor.

Further autonomic options could evolve from the environment knowledge gained by the tool; for example, the ability to spot a process running intermittently or unstably. It may have a history of failing after running for a certain period of time on some executions. In such a case, the process (or the application) may need re-configuration or re-installation in order to run smoothly, in effect providing options for self-optimizing and in so doing preventing the system degrading further which indirectly provides self-protection also. The self-healing tool provides various facilities to the programmers to make their programs autonomic and which could recover from the failure. This also helps to move some particular stage of program after restart saving the time and cost.

The research work presents a novel approach for the concept of self healing tool in client server mode. The future work is to add more sophisticated healing functions and methods to the health monitor and making it as the powerful self healing tool. The work also has also to be resumed in the further implementation of the external monitor and main monitor. This self healing tool can play a crucial role in the applications of large scale as this is based on client server technology. The advantage of client server technology fetches a lot to this self healing tool. This can be very much useful in the applications of autonomic computing that will be emerging in future. Database systems, server, web are some of few examples where this technology can be applicable and ultimately it seems a great thought to make systems take their own care.

REFERENCES

[1] IBM, " Autonomic computing: IBM's perspective on the state of informationtechnology;"http://www-1.ibm.com/industries/government /doc/ content/resource/thought/2786061.html

[2] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Systems Journal*, Vol 42, No 1, 2003

[3] IBM, " An architectural blueprint for autonomic computing," April 2003

[4] R. Sterritt and D. Bustard, " Autonomic computing means of achieving dependability?", *Proceedings IEEE International Conference On the Engineering of Computer BasedSystems (ECB,03), Huntsville, Alabama, USA*, April 7-11 2003

[5] D. Garlan, J. Kramer, and A. Wolf , An architectural support for self-adaptive software for treating faults proceedings workshop on self-healing systems (WOSS'02), ACM Press, Charleston, South Carolina, Nov. '02

[6] Horn, Paul , " Autonomic computing IBM's perspective on the state of informationtechnology," available from the IBM corporation at http:// www.research.ib.com /autonomic/ manifesto/autonomic computing.pdf

[7] Jeffrey O. Kephart and David M. Chess , " The vision of autonomic computing ," *IEEE Computer Society*, pp. 41-50, Jan 2003

[8] D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni A. Mohindra , D. G.Shea, and M. Vanover, " Autonomic personal computing", *IBM Systems Journal* , Vol 42,No 1, 2003

[9] Roy Sterritt and Saulai Chung, "Personal autonomic computing self-healing tool," *Proceedings of the 11th IEEE International Conference and workshop on the Enggineering of Computer Based Systems* (ECBS'04) 2004

[10] WhatisP2P,http://compnetworking.about.com/library/products/weekly /a093000a.html

[11] Peer-to-PeerComputingisGoodBusiness,http:// www.intel.com/eBusiness/products/peertopeer/ar010102.html

[12] Peer-to-Peer to Architecture, http:// 80211-planet.webopedia.com /peer_to_peer_architecture.html

[13] ClientServerArchitecture, http://en.wikipedia.org/wiki/Client-server

[14] http://www.webopedia.com/TERM/C/client_server_architecture.html

[15] Client Server vs Peer to Peer,

[16] http://www.dewassoc.com/support/networking/serverpeer.htm

Ms. N. RukmaRekha was born on 01-05-1982 in AndhraPradesh, India. She obtained her B.Tech, M.Tech degrees from Andhra University in 2003 and 2005 respectively. Currently, she is pursuing her Ph.d as part-time from the same university. She has around 6 years of teaching experience. She has the experience of guiding around 16 Post graduate students for their M.Tech/M.C.A Project Thesis. Ms. Rukma Rekha is now Assistant professor from Dept.of Computer and Information Sciences in University of Hyderabad.

Prof. Maddali Surendra Prasad Babu was born on 12-08-1956 in Prakasam district of Andhra Pradesh. He obtained his B. Sc, M.Sc and M. Phil and Ph.D. degrees from Andhra University in 1976, 1978, 1981and 1986 respectively. During his 27 years of experience in teaching and research, he attended about 28 National and International Conferences/ Seminars in India and contributed about 33 papers either in journals or in National and International conferences/ seminars. Prof. M.S. Prasad Babu has guided 98 student dissertations of B.E., B. Tech. M.Tech. & Ph.Ds. Prof Babu is now Professor in the Department of Computer Science & Systems Engineering of Andra University College of Engineering, Andhra University, Visakhapatnam.

Prof. M.Surendra Prasad Babu received the ISCA Young Scientist Award at the 73rd Indian Science Congress in 1986 from the hands of late Prime Minister Shri Rajiv Gandhi.

Prof. Babu conducted the proceedings of the Section of Information and Communication & Sciences and Technology including computer Science of the 94th Indian Science Congress as a president of that section in January 2007. Prof. Babu was also the sectional committee member of the Indian Science Congress in the sections of Mathematics and ICT in 1988 and 2005 respectively. He is also sectional secretary for the section of Information and Communication & Sciences and Technology of Indian Science Congress.