PACT: A new approach to self healing mechanism (Autonomic Computing)

Prof. V.K. Pachghare
Lecturer, Computer Engineering & IT Department,
College of Engineering, Pune, (An autonomous Institute of Government of Maharashtra, India)

Summary
Self managing and autonomic computing are emerging as new significant trend in design of computer systems. As we try to make the systems as autonomous as possible, new problems and new solutions emerge. The purpose of this paper is to consider the problem of self healing system and to suggest some solutions. We need systems which can detect their failure and could not only recover from it but also should be able to resume their work from the point of failure. This paper will talk about one such new approach towards this problem. It talks about an experimental framework created to build such applications. The programs written using this framework will be able to save their states and could start working after restart from the point it was stuck.

1. Introduction

Computer systems are becoming more demanding, complex and challenging. 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.

Organizations that develop complex computer systems are facing additional market conditions such as demands for more functionality, ever decreasing time-to-market, high employment costs, leading to a requirement to utilize only semi-skilled employees. The other factor that needs to be considered is that the performance of these systems can never be compromised and they are never expected to fail. But what happens if those systems go down for some reason and the processing ever done is lost. This may result in big loss to an organization. (e.g. A banking software reading data from mainframe and calculating total deposit, interest and loss or profit of the bank. And at crucial point of time, program goes down and needs to be restarted. The cost involved in reacquiring the data will be huge.)

2. Need of autonomic computing

Simply stated from above, managing complex systems has grown too costly and prone to error. People under such pressure make mistakes, increasing the potential of system outages with a concurrent impact on business. The following points will reveal more about need of autonomic computing.

It is now estimated that one-third to one-half of a company's total IT budget is spent preventing or recovering from crashes.

- Nick Tabellion, CTO of Fujitsu Softek, said: "The commonly used number is: For every dollar to purchase storage, you spend $9 to have someone manage it."
- 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, administrator salaries, and service and support from database suppliers).
- When you examine data on the root cause of computer system outages, you find that about 40 percent are caused by operator error, and the reason is not because operators are not well-trained or do not have the right capabilities. Rather, it is because the complexities of today's computer systems are too difficult to understand, and IT operators and managers are under pressure to make decisions about problems in seconds.
- A Yankee Group report estimated that downtime caused by security incidents cost as much as $4,500,000 per hour for brokerages and $2,600,000 for banking firms.
- David J. Clancy, chief of the Computational Sciences Division at the NASA Ames Research Center, underscored the problem of the increasing systems complexity issues: "Forty percent of the group's software work is devoted to test," he said, and added, "As the range of behavior of a system grows, the test problem grows exponentially."

In a survey made on causes of outrages in four areas, most frequently found outrages are:

For systems: operational error, user error, third party software error, internally developed software problem, inadequate change control, lack of automated processes.

- For networks: performance overload, peak load problems, insufficient bandwidth.
- For database: out of disk space, log file full, performance overload.

Manuscript received October 5, 2007
Manuscript revised October 20, 2007

● For applications: application error, inadequate change control, operational error, non automated application exceptions.

This results in the need for self-managing systems and new development approaches that can deal with real-life complexity and uncertainty. The challenge is to produce practical methodologies and techniques for the development of such self-managing systems, so that they may be leveraged to deal with failure and recover easily.

3. What is autonomic computing?

Autonomic computing can be seen as a holistic vision that enables a computing system to "deliver much more automation than the sum of its individually self-managed parts". A system is considered a collection of computing resources working together to perform a specific set of functions.

While the definition of autonomic computing will likely transform as contributing technologies mature, the following list suggests eight defining characteristics of an autonomic system.

● To be autonomic, a system needs to "know itself"—and consist of components that also possess a system identity.
● An autonomic system must configure and reconfigure itself under varying and unpredictable conditions.
● An autonomic system never settles for the status quo—it always looks for ways to optimize its workings.
● An autonomic system must perform something akin to healing—it must be able to recover from routine and extraordinary events that might cause some parts to malfunction.
● A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection.
● An autonomic computing system knows its environment and the context surrounding its activity, and acts accordingly.
● An autonomic system cannot exist in a hermetic environment (and must adhere to open standards).
● Perhaps most critical for the user, an autonomic computing system will anticipate the optimized resources needed to meet a user's information needs while keeping its complexity hidden.

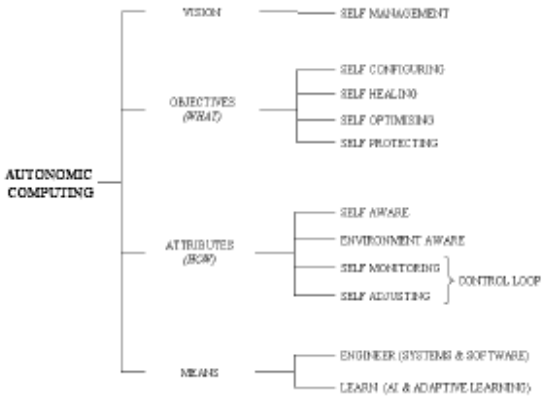

Fig. 1 Autonomic Computing Tree

The standard PACT is designed by considering almost all these characteristics. But before the PACT gets explained, there are some fundamentals about autonomic computing those should be overlooked and understand the category of the standard.

Self configuring: The system environment keeps changing regularly, so it is required that it should be able to adjust itself to the particular conditions and act "*wisely"*. When software and hardware systems have the ability to define themselves "*on the fly"*, they are self configuring. This aspect allows system to install new feature as requirement with no disruption of services. Systems must be designed to provide this aspect at a feature level with capabilities such as plug and play devices, configuration setup wizards, and wireless server management. And all this should be done without or minimum human intervention. The goal of autonomic computing is to provide self-configuration capabilities for the entire IT infrastructure, not just individual servers, software, and storage devices.

Self healing: This is the focus of discussion in this paper. PACT is basically self healing mechanism dealing with processes. Systems discover, diagnose, 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 line, fixing or isolating the failed component, and reintroducing the fixed or replacement component into service without any apparent application disruption. Systems will 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 design of each hardware and software product

toward continuous availability. The fig.1 represents all fundamental self properties of autonomic system.

Self optimizing: Systems monitor and tune resources automatically. Self-optimization requires hardware and software systems to efficiently maximize resource utilization to meet end-user needs without human intervention. The systems may include industry leading technologies such as logical partitioning, dynamic workload management, and dynamic server clustering. These kinds of capabilities should be extended across multiple heterogeneous systems to provide a single collection of computing resources that could be managed by a "logical" workload manager across the enterprise. Resource allocation and workload management must allow dynamic redistribution of workloads to systems that have the necessary resources to meet workload requirements.
Similarly, storage, databases, networks, and other resources must be continually tuned to enable efficient operations even in unpredictable environments. Features must be introduced to allow the enterprise to optimize resource usage across the collection of systems within their infrastructure, while also maintaining their flexibility to meet the ever-changing needs of the enterprise.

Fig.2.Four fundamental "self" of autonomic computing

Self protecting: Systems anticipate, detect, identify, and protect themselves from attacks from anywhere. Self-protecting systems must have the ability to define and manage user access to all computing resources within the enterprise, to protect against unauthorized resource access, to detect intrusions and report and prevent these activities as they occur, and to provide backup and recovery capabilities that are as secure as the original resource management systems. Systems will need to build on top of a number of core security technologies already available today, including LDAP (Lightweight Directory Access Protocol), Kerberos, hardware encryption, and SSL (Secure Socket Layer). Capabilities must be provided to more easily understand and handle user identities in various contexts, removing the burden from administrators.

4. Need of industry standard

Most IT infrastructures are composed of components supplied by different vendors. Open industry standards are the key to the construction of autonomic computing systems. Systems will need more standardization to introduce a uniform approach to instrumentation and data collection, dynamic configuration, and operation. Uniformity will allow the intersystem exchange of instrumentation and control information to create the basis for collaboration and autonomic behavior among heterogeneous systems.
For example, in storage systems, a standard that has been proposed for specifying data collection items is the Bluefin specification. Bluefin defines a language and schema that allow users to reliably identify, classify, monitor, and control the physical and logical devices in storage area networking. The Storage
Networking Industry Association (SNIA) has taken this standard to the Distributed Management Task Force (DMTF). SNIA is using Bluefin as the basis for its storage management initiative, the intent of which is to become the SNIA standard for management.

5. PACT

The standard provides user with APIs to keep track of state of program and data being used in the program. It is basically dependant on client server architecture and uses RPC for communication. The client is the program using the functions provided and server is the program keeping track of all requested processes. Server also does work of housekeeping the critical data of program requested by user. User is allowed to save his data to server so in case of failure and restarting the work, his process won't be required to beg for the same data again to the source.

A new concept of heartbeat is introduced here and is used for tracking status of application. As a human is said to alive if his heart is working i.e. heartbeats can be listened. The programmer can decide what should be the period between two heartbeats. The server listens to the heartbeat sent by the program. In this way server can predict which process is alive or working correctly and which one is dead or hanged or its complete system is down. The work of server in this situation is very clear, it should make process to recover from the trauma. The obvious way is to restart the process. But this won't be a good solution if everything (calculation and acquisition of

data) that has been done by process need to be done again. But fortunately the PACT system is capable of storing the data of process along with its versions. So after restart, the program can just check whether it was registered and could regain the data and version. It is up to process now what to do with data and version. If programmer uses this wisely, the program will be able to start executing from the point of latest version i.e. the point which sent the data with that version number for saving.
The restarting of process on remote machine is achieved using restart server on that machine. It's a small RPC server program which is contacted by PACT server. A command is sent by PACT server to this restart server and this command is executed by this server.

5.1 Functions Explained

Let's look into more detail from the programmer's perspective towards this standard. First of all the program should be registered for the service. This can be done using the function provided. The arguments required for this are command line arguments, server name and the number of versions you want to save. This is because; it may be expensive to save all data. This will look like:

int pact_register(int argc,char *argv[],int states, char * savior);
Description : This is the main api of the whole pact structure the pact_register is the main register call which asks pact to check if the application data actually present at the server daemon. if it is then the application can actually begin processing from respective state.
Return value :
pact_register returns the version number of the latest version of data available. Else it returns -1 for first time registration and the application gets registered at the server daemon.
int pact_deregister();
Description : pact_deregister removes any registry at the PACT as the client has finished with its work and can leave the service the PACT has offered to it.
Return value :
1 returned on successful deregistration
-1 if any error occurred while performing deregistration
int pact_senddata(char * pact_data , int version);
Description : pact_senddata is basically used as a wrapper which collects the whole data user wants to send to the pact server. And do the processing of saving the data with latest version present as accordingly. The user just needs to bind all the data which he/she wants to send as in a pact_data string and perform the send operation.
Send data is mainly used when application covers certain amount of processing and having a safe stage in mean time.
Return value :
returns -1 if some error occurs else returns 1 on success.
char * pact_receivedata();
Description : pact_receivedata grabs the latest version of the data available at the server daemon and pass it to the application running return value it returns the latest version data in terms of a character string and application can then collect the data out of that string.
Return value :
Pact_receivedata() returns the latest version of data in terms of raw string. Else returns -1 on error.

int pact_startmonitor(int wait);
Description :pact_startmonitor starts the monitoring performed by PACT on the client program over the particular time period specified as a wait argument. the API should be called before the start of processing with approximate time that it could take to finish.
Return value :
-1 returned on error otherwise returns 1 on success.
int pact_sendbeat()
description : pact_sendbeat checks the successful processing and informs PACT to stop the watch made. i.e. it notifies the PACT that the client done with the processing in specified time without any occurrence of error.
Return value :
Returns 1 on success otherwise -1 if any error occurred during call made

5.2 Ideal approach of use

It is always great to have useful services. But it is up to the programmer's ability and creativity to use those. The same thing applies here also.

The first thing that should be done ideally is to register in the early stage of process. It is highly recommended that programmer make a kind of checkpoint in their programs. And at those points, the important data can be saved. It is also very critical to decide how much checkpoints are needed and where to create those checkpoints.

When process is restarted, it will receive the version number at the point of registration. Now it is very necessary to decide the next strategy from this point. Does program need to go to last saved state or to some other stage and carry on the work. The other option will be to get the data and create a log of processing that may have led to failure. This will be helpful to figure out the mistakes done by programmers.

When the program is at a stage where it may wait for a long while may be waiting for user to feed something through console, monitoring should be stopped. Otherwise there is possibility that it will be restarted.

5.3 Applying 8 characteristics

As stated above, there are 8 characteristics of autonomous system. Those can be applied to application built using PACT.

According to 1st property, PACT knows itself by keeping eye on its environment and takes decision based on it.

The second property i.e. Self configuring holds here. The system monitors the CPU usage at time of restart on client machine. If it is too high, the process should be given another chance. So restarting is delayed by some time.

Third property also holds true. As the system is continuously monitoring client, it does not just accept the fact of failure.

As explained above, the project is about self healing. So forth property is also valid.

The communication can be carried out in secured manner if user wants. So it solves problem of security.

The fifth property is similar to 2nd. It keeps an eye of resource usage and modifies its behavior.

The system uses RPC calls internally. So it adheres to open standard satisfying 7th characteristic.

5.4 Process migration

Another one interesting aspect that we should consider here is the process migration capabilities of system. There can be a case when the host machine fails due to some reasons like power failure. But we have the data saved with us. So it would be great if the process is shifted to some other machine(say savior machine) which has exact same copy of application. We can configure the system by providing savior's address at the time of registration and could actually migrate the process if host machine is dead. So when the PACT server knows about the failure of the host machine, it will contact with savior machine and the process could be started from the point where host machine was crashed. This will make the system immune to even power failures.

6. Systems evolving towards autonomic behavior

There can be many applications of the explained PACT standard. Here we are considering some areas where industries can apply those standards and create autonomic systems.

1. Database systems: The autonomous database software will use statistics, analyze it and learn from historical system performance. The tools will help to automatically detect potential bottlenecks and avoid them before they come or overcome those faults without having to start everything again.
2. Web servers and software: Once improved instrumentation is available, autonomic functions can be introduced that enable the Web server infrastructure to automatically monitor, analyze, and fix performance problems. As an example, suppose an application server is freezing-up intermittently, and no customer transactions are being processed for several seconds, thus losing thousands of dollars in business, as well as customer confidence and loyalty. Various techniques like real time monitoring, auto tuning can be used to anticipate the freeze-up is before it happens. The interesting feature of saving data in PACT can be used to save data about the connections. So even if server goes down and restarted, sessions can be restored without any loss.
3. Servers: Computers can be built that need less human supervision. Computers can try to fix themselves in the event of a failure, protect themselves from hacker attacks, and configure themselves when adding new features. The fact that there is facility to monitor those and save their states is very useful. Servers are critical points in any modern systems. PACT can provide that critical operation to keep the servers up. The server program can be designed with the standards and some other machine can monitor it. The other use can be interpreted as to keep the log record and check those after recovery. Servers can use software algorithms that learn patterns in Internet traffic or application usage, and provision resources in a way that gives the shortest response time to the task with the highest business priority. Server support for heterogeneous and enterprise workload management, dynamic clustering, dynamic partitioning, improved setup wizards, improved user authentication, directory integration, and other tools to protect access to network resources are all steps toward more autonomic functioning.

7. Conclusion

Autonomic computing is emerging approach towards future systems and software. These systems will be able to imply all 4 "self" factors and will minimize the human interaction for very ordinary decisions. The characteristics of autonomic computing were discussed.

The need of standards in software development to support self healing was focused and a standard PACT was discussed under this category. It provides various facilities to programmers to make their programs autonomic and which could recover from the failure. This also helps to move to some particular stage of program after restart saving the time and cost.

The applications of autonomic computing are many and will emerge a few more in future. Database systems, servers, web are some of few examples. And ultimately it seems a great thought to make systems take their own care.

References

[1] Roy Sterritt, Mike Hinchey "Autonomicity- A antidote for complexity?"
[2] A. G. Ganek, T. A. Corbi, "The dawning of autonomic computing era" IBM systems journal, vol 42, no 1, 2003.
[3] White Paper
Predictive Self-Healing in the Solaris™ 10 Operating System.
[4] "Autonomic Computing. Powering your business for success" IBM white paper on autonomic computing 2005.
[5] Jana Koehler, Chris Giblin, Dieter Gantenbein, Rainer Hauser, IBM Zurich Research Laboratory "On Autonomic computing architecture"
[6] Autonomic computing Overview from http://www.research.ibm.com/autonomic/overview