

# AMATA: Software Architecture and Implementation of High Availability Support for Beowulf Cluster

*Jullawadee Maneesilp, Putchong Uthayopas  
Parallel Research Group, CONSYL  
Department of Computer Engineering, Faculty of Engineering,  
Kasetsart University, Bangkok, Thailand  
Phone: (662) 942-8555 Ext. 1416  
Email: {g4265082, pu}@ku.ac.th*

---

**ABSTRACT** -- High-availability support for Beowulf cluster becomes a critical factor in the acceptance of this platform for mission critical use in enterprise environment. A well defined and extensible HA software architecture is needed. This paper presents the proposed high-availability software architecture called AMATA. AMATA architecture clearly defines the software component and interaction for High Availability support. Many cases such as system software failure, registered user software error, hardware mal-function, and hardware overload can be detected and handle in a systematic way. Both discovery and recovery process can be added to provide an intelligent and automatic fault recovery process. Currently, a prototype implementation has been developed and the results obtained from that implementation have also been included.

**KEY WORDS** -- High availability, Beowulf clusters, Fault Tolerance,

---

## 1. Introduction and Motivation

Beowulf cluster [1] has been widely used as a scalable information server or large scientific parallel computer. Many important software and algorithm have been successful ported to this platform. Nevertheless, operating large Beowulf cluster system reliably is still a problem since most of the commodity off-the-shelf parts are not initially designed as an integral part of the highly reliable systems. PC components are less reliability than commercial server system. This problem is the main obstacle in using the cluster system for mission critical task in enterprise or industrial computing. In the commercial Unix marketplace, high availability [2] is today a key to selling server solution and virtually every Unix suppliers have their own HA software solution for customers. However, there is still a need for powerful open source software support that detects and recover from fault that allows users of Beowulf systems to construct a cost effective HA server solution for their computing needs.

In this paper, we present our high availability model and implementation for Beowulf cluster environment called AMATA. AMATA architecture define a well structure software architecture and interaction between software components for HA support in Beowulf systems. Under the framework of AMATA, software components can be built to detect major systems fault and recovery from that fault in a systematic way. Moreover, user can easily add some intelligence logic to the system to automate the detection and recovery processes.

The organization of this paper is as follows. Section 2 presents the discussion about some related works. Then, we explain briefly about our background technology that involves this work in section 3. Next in section 4 we will discuss about the design and implementation of AMATA system. Finally, we give the conclusion in section 5.

## 2. Background and Related Work

High-availability (HA) software can be classified into two main approaches. The first approach is to extend HA service in kernel level. Although, this approach can be very efficient, the portable implementation is not possible. Also, it is difficult to keep pace with the rapid kernel changes that happened with Linux kernel. Example of such work is Piranha [3] and Solaris-MC [4]. Solaris-MC is a prototype operating system for Solaris cluster that provides a single system image and high-availability by extending operating system abstraction across the cluster.

The second approach is to extend the high-availability service in user space. Not only does this approach has a higher portability, but also reduce the need to frequently release new kernel patches. One example is Keep alive project, which is based on VRRPd [5] implementation of VRRPv2. VRRP is a standard protocol that helps elect a master server on LAN when the old master server fail. However, VRRPd only support the high-availability in case of server fails. No solutions for user services or system services fail are provided yet. Linux FailSafe [6] is a community development effort lead by SuSE and SGI to port SGI IRIS FailSafe product to Linux. FailSafe provides a full suite of high-availability capabilities for Linux. These include full N-node

cluster membership and quorum services application monitoring and failover-restart capabilities, with a set of GUI tools for administering and monitoring HA clusters. Unfortunately, only plug-in of Linux FailSafe is free. Hence, by delivering a fully available open source cluster, we can stimulate the wide spread use of Beowulf clusters as a solution for mission critical IT needs.

### 3. AMATA Architecture

#### 3.1 AMATA Structure

The proposed AMATA HA architecture is illustrated in Figure 1. AMATA architecture consists of the following components.

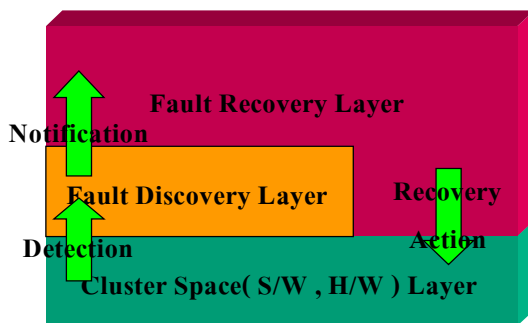


Figure 1. AMATA Architecture

**Cluster Space Layer** is an abstract view of cluster systems. This layer consists of 2 classed of objects, namely, *software objects* and *hardware objects*. Examples of software objects are tasks, OS services, daemon, and user processes. On the other hand, hardware object is node, interface card, network, and hard disk.

Each object will have an attached component called *object probe* that monitors the object performance information and provides an interface for external object to access these information. Performance parameter of each object is represented by a set of counters such as CPU usage, numbers of packet send/receive and state. Currently, three types of object probe are available. Firstly, **Hardware object probe** is a module that connects with the hardware monitoring system to receive hardware information such as CPU utilization, memory usage and network traffic. Secondly, **System services object probe** is a module that monitors the system services on each node. Thirdly, **User service object probe** is used to monitor users applications.

**Fault Discovery Layer** is a layer receives the information from Object Probe, discover the potential fault and notify the recovery layer using event mechanism. The discovering process in this layer is based on a set of rules called *Discovery rule*. The role of theses rules is to transform a set of data received from cluster space into a set of answers. The execution of these rules takes place in part of the code called *discovery module*.

**Fault Recovery Layer** this layer receive event form fault discovery layer to generate action to be taken and consists of recovery rule, recovery module and recovery action driver.

The event that received from Event Service in KSIX[8,9] will pass to each event handler. We provide default action for each known event implemented with python. User can also extend their recovery rule easily by write python script follow by exist scripts.

#### 3.2 Automatic Fault Detection and Recover Process

When any fault happened in the system, the process of automatic fault detection and recovery can be explained as a flowchart shown in Figure 2.

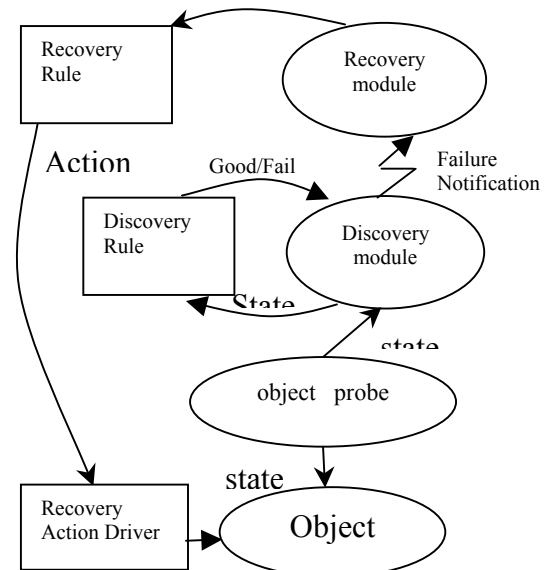


Figure 2. Recovery process

In Figure 2, the step started from system object in cluster space. Each object will be probed periodically to measure its performance state by discovery module. This state will be passed to internal discovery module, which transform it to a set of decision such as good or fail. If the failure is detected, a failure notification event will be sent to recover module. Recover module will determine an appropriate course of action using information embedded in the event and recovery rule. The action will be send to a code called recovery action driver which execute series of commands that automatically solved the problem for the user. In this process, users can add many complex and intelligent logic to system later to allow very automatic fix of the problems in cluster system.

#### 3.3 AMATA Implementation

AMATA is implemented as a set of daemon and script that execute on each node in the system. The AMATA implementations rely on many services provided by KSIX [8] Middleware. After user boot this middleware, KSIX will automatically load AMATA as one of its services. The structure of this software system are as shown in Figure 2.

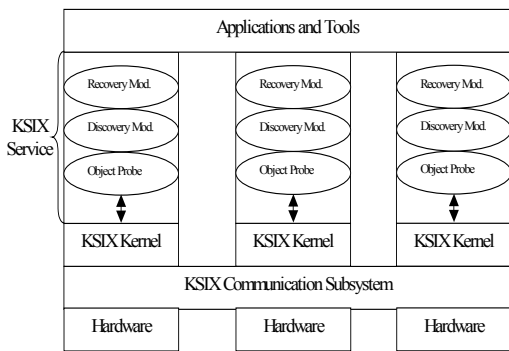


Figure 3. AMATA After KSIX boot

Object probe, discovery module and recovery module will use KSIX event service to communicate to each other. Moreover, KSIX feature called automatic restart process has been employ so that AMATA system will be automatically restarted when it fail.

One daemon called AMATA console, is used to log the notification from the fault recovery module. Once the notification has been accepted, the console daemon will start some predefined script corresponded to that event. This feature allows user to hook some logic to report error. For instance, having a script that send error message using the phone paging mechanism. User can also control this daemon by opening a socket connection to it and communicate with a simple command. The previously mentioned operation can be summarized as illustrated in Figure 3.

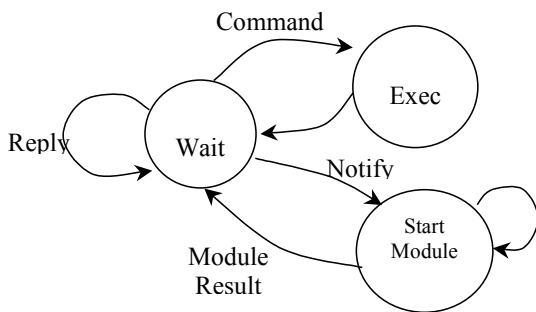


Figure 4. Console process diagram

### 4. Experimental Result

The cluster system used for the test consists of 4 Athlon 1 GHz and three Athlon 950 MHz with 512 Mbytes of memory per machines. These machines run Linux RedHat 6.2. They connected together with Myrinet Switch and Ethernet switch.

In the experiment, the purpose is to measure the time used to recovery from fault. In this test, we tested the system responds time in three cases. In each case, there are three services that are recovered from the artificial fault. For the first, case, we measure the recovery time when the system has no load. For the second test, a system is load with computation task. This computation load is generated from Linpack benchmark program at problem size 200. Finally, we test the system under I/O load condition. This I/O load has

been simulated using the reading and writing to disk file. The elapse time is then measured starting from the termination time of the services until the time that system service has been fully recovered. The results are as reported in Table 2 and the CPU utilization has been reported in Table 3. From the results, we can see that the recover process has happened very fast. The implementation consume very low CPU usage which demonstrate that the implementation is quite efficient. The increase in I/O and CPU load do effect the recover speed but the clear result of the impact will need more study.

Table 1. Recovery Time

No. of services	No Load (sec)	CPU Load (sec)	I/O Load (sec)
1	3.960	12.782	6.840
2	4.061	19.056	17.760
3	4.794	19.458	16.249

Table 2. Percentage CPU of utilization

	No Load	CPU Load	I/O Load
%CPU of Utilization	1.5-3	0.5-1	1.5-2

### 5. Conclusion

The contribution of this work is mainly the new proposed concept well define architecture for HA on Linux cluster that provides a basis for real working implementation. In this model, we clearly define a modular and scalable, HA model that is easy to extend. The future work will include a better and broader implementation of each part, the addition of more intelligent logic that allows better and more automatic discover and recover of fault, the prediction of potential failure, and more study on performance and impact of external factors to the recovery process.

### 6. References

- [1] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. E. Packer, "Beowulf: A Parallel Workstation for Scientific Computation", In Proceedings of the International Conference on Parallel Processing 95, 1995.
- [2] Harald, M, "Linux High Availability Howto", Free software Foundation, Inc., Boston, Massachusetts, USA. 1998.
- [3] Kavin Railsback, "Linux Clustering in depth", Linux Magazine. August, 2000.

- 
- [4] Yousef A. Khaladi, Jose M Bernabeu, Vlada Matena, Ken Sherriff and Moti Thadanai, "Solaris MC: A multi computer OS", In the proceeding of USENIX 1996 Annual Technical Conference, San Diego, California, January 1996.
- [5] Jerome Etienne, "High availability: vrrpd, usage, configuration and security", In the proceeding of Ottawa Linux Symposium 2001, Ottawa, Canada, July 24-28, 2001
- [6] Lars Marowsky-Bree, "Linux FailSafe –High Availability for Linux", In the proceedings of Ottawa Linux Symposium 2000, Ottawa, Canada, July 19-22, 2000.
- [7] Putchong Uthayopas, Jullawadee Maneesilp, Paricha Ingongnam, "SCMS: An Integrated Cluster Management Tool for Beowulf Cluster System", Proceedings of the International Conference on Parallel and Distributed Proceeding Techniques and Applications 2000 ( PDPTA'2000) , Las Vegas, Nevada , USA , 26-28 June 2000.
- [8] Thara Angskun, Putchong Uthayopas and Choopan Ratanpocha, "KSIX parallel programming environment for Beowulf Cluster", Technical Session Cluster Computing Technologies, Environments and Applications ( CC-TEA ), International Conference on Parallel and Distributed Proceeding Techniques and Applications 2000 ( PDPTA'2000), Las Vegas, Nevada , USA, June 2000.