A Comprehensive Monitor Model for Self-healing Systems

Dong Xikun,
College of Computer Science and Technology, HEU
Harbin, China
dongxikun@hrbeu.edu.cn

Wang Huiqiang,
College of Computer Science and Technology, HEU
Harbin, China
hqwang@hrbeu.edu.cn

Lv Hongwu,
College of Computer Science and Technology, HEU
Harbin, China
lvhongwu@hrbeu.edu.cn

***Abstract*—as monitor plays a critical role in self-healing loop as the basis of self-healing, this paper proposes a comprehensive monitor model for self-healing systems on the basis of building model for general applications and organically combining monitor methods for different attributes of files. The monitor model can improve the performance of self-healing by reducing the grain size of monitor and recovery. And the experiment results show that the proposed monitor model has better monitor performance and lower missed rate than single aspect monitor and it can decrease performance cost of subsequent recovery.**

Key words—monitor; comprehensive monitor model; application model, self-healing

The heterogeneous nature, complexity of modern software systems, and the wicked environment they are working in inevitably cause software failure, system halted, error results and so on, which lead to serious consequence, such as huge economic loss. However, the manual management and maintenance of the software systems become more and more difficult. As a result, a more flexible self-recovery method is expected to solve these problems, which made the emergence of self-healing technique. Self-healing is a sub-topic of autonomic computing, whose main target is to realize autonomic management of recovery and to deduce human intervention. The idea is enlightened by the human autonomic nervous system which can manage regional function of the body automatically without control of cerebral consciousness[1]. Correspondently, the self-healing mechanisms in computer system provide maintaining and resuming abilities in a stated condition automatically by designing the system reasonably.

The structure of self-healing is similar to the autonomic unit in Figure 1 proposed by IBM. It's an MAPE loop with feedback which is composed of monitor, analyzer, planner and executor. The MAPE loop manages systems automatically by simulating the control loop of human autonomic nervous system. Different implementations of self-healing have different structures, but they all follow the MAPE loop more or less. Being the basis of other components, monitor is essential in the four parts of MAPE loop. Kephart points out that monitor is a key feature of autonomic element, and it's responsible for collecting the information of component's responding time[1], files' dependability, and resources occupied by the component, etc.

The self-healing work flow starts from monitor, and all input information of other components is obtained from monitor directly or indirectly. Analyzer and planner are even omitted in some simple implementations, but monitor is always an indispensable module. Therefore, monitor is an essential part of the self-healing structure and a necessary means to obtain healthy information of the system.

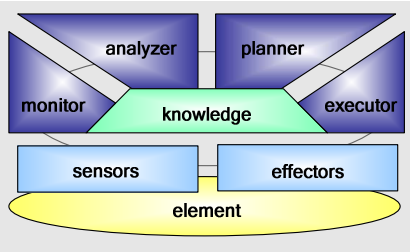

Figure 1 Autonomic unit

1 Related work

The concept of monitor exists in many fields such as military, business, and daily life. For example, military satellites, cameras in security systems, and fire alarms are all some kinds of monitor. To get and analyze the system information, the concept of monitor has been already introduced into information techniques for decades. Monitors exist everywhere in the information systems because all means to obtain information from the system can be regarded as monitor, like files tamper monitor[2], network failure monitor[3], software fault monitor[4] and

978-0-7695-4258-4/10 $26.00 © 2010 IEEE
DOI 10.1109/MINES.2010.159

so on.

Different monitors have different implementations and targets. Classified according to the means of obtaining information, monitor has three types: active monitor, passive monitor and mixed monitor.

Active monitor: these monitors monitor target systems actively, namely, monitoring action is launched by the monitor. For example, in the self-healing framework PANACEA[5] proposed by IBM, there is a responding-time monitor which is responsible for monitoring whether an overtime response of the component occurs or not, which is a kind of active monitor. The polling monitor, algorithm, [6] which is often adopt in distribute network, is a kind of active monitor too.

Passive monitor: the initiative of monitoring is handled by the monitored side which launches monitoring action. For example, in the framework proposed in [7], the information of monitored system is sent to administrator within running time by the probes which are injected into system in construction phase through Monitor Requirements Language(MRL).

Mixed monitor: it's a combination of active monitor and passive monitor. Paper [8] proposes a mixed monitor algorithm based on heartbeat protocol as active detecting means and attendant message protocol as the passive means.

Nowadays, there are a lot of mature computer monitor software and prototype systems like the monitor prototype system InteMon[10][11] designed by Carnegie Mellon University, the Parmon system[12][13] developed by the University of Melbourne, the monitor software Ganglia[14] developed by UC Berkley, server monitor software bWatch[15] which is using in Beowulf Cluster constructed by Paralogic company, the network management software OpenView[16] developed by HP, the performance monitor model CPMM[17] designed by Tsing Hua University, the network monitor model proposed by University of Sichuan, the GridMon[19], Octopus[20] system and etc.

In this paper, general applications are taken as target systems and we build Application Model(AM) for them at first, then Comprehensive Monitor Model(CMM) towards self-healing are proposed based on AM.

2. Comprehensive monitor model(CMM)

2.1 Build Application Model(AM)

When an application is not running, all files are stored in hard disk. After the application starts, some files are loaded into the memory while others are still in hard disk. So files in the application have two states: hard disk state(H-State) and memory state(M-State). Therefore, the files in an application are divided into two types: one type only have H-State, which is called H-State files while the other type has both H-State and M-State, which are called H&M-State files. For example, class files are loaded into memory when they're instantiated, so they have both of the H-State and M-State. But some configuration files can only be read by some special threads, so they only have H-State.

Every file in applications has its natural attributes. The attributes related to the H-State is called Static Attributes(SA). For example, from the perspective of security, the above-mentioned configuration file has consistency including parameters like modified time, file size and hash value, etc. The H&M-State files have not only SA but also Dynamic Attributes(DA) related to M-State. And DA is divided into Inner Attributes(IA) and Outer Attributes(OA). IA are the attributes representing the nature of files in the memory, like efficiency including parameters such as responding time and etc; running cost including parameters such as CPU occupancy rate, memory occupancy rate, numbers of threads, numbers of loaded classes and so on. While the application is running, the files in memory communicate with each other. Hence the files have attributes related to communication called OA. It includes parameters like source component, target component, normal communication time limits, maximal communication time limits and etc.

AM: an application can be formally presented as $App=\{file_1,file_2,\ldots\}$, $file_i$ is the file in the system which can be formally presented as a triple $file=(S,SA,DA)$, S represents state, $S=\{H\text{-}state, H\&M\text{-}state\}$; SA represents static attributes, $SA=\{attr_1^s,attr_2^s,\ldots\}$, where $attr_i^s=\{para_{i1}^s,para_{i2}^s,\ldots\}$ concludes some parameters; dynamic attributes $DA=\{IA,OA\}$, inner attributes $IA=\{attr_1^i,attr_2^i,\ldots\}$, and $attr_i^i=\{para_{i1}^i,para_{i2}^i,\ldots\}$; outer attributes $OA=\{attr_1^o,attr_2^o,\ldots\}$, and

$attr_i^o=\{para_{i1}^o,para_{i2}^o,...\}$。

2.2 CMM

Monitor model is the abstraction of process and the method for monitoring an application and it determines the work flow and implementation of monitor.

CMM: on the basis of building AM, CMM can be formally presented as a quadruplet（*App,* Ψ*, t, Events*）. App is the target application, and Ψ represents the set of attributes monitor method. Events are results of monitoring, and t is the time. And the mappling Ψ: $App\times t\rightarrow Events$.

According to different target or functioning scope, attributes' monitor methods are divided into Static Attributes Monitor Method(SAMMethod), dynamical Inner Attributes Monitor Method(IAMMethod) and dynamical Outer Attributes Monitor Method(OAMMethod). They monitor SA, IA and OA respectively. Attributes' monitor methods can be presented as $\Psi=\{SAMMethod, IAMMethod, OAMMethod\}$.

Theorem: for some accurate time t, for any $application=\{file_1,file_2,...\}$, $\exists\ m\subseteq\Psi$, $\psi(applacation,t)=\psi(\bigcup_k file_k,t)=\psi(\bigcup_h\bigcup_i\bigcup_j para_{ij}^h,t)=events$, $h\in\{s,i,o\}$, and *events* are subset of *Event.*

The structure of CMM is shown as Figure 2, in which the dashed line represents monitoring data flow and the solid line represents business data flow.

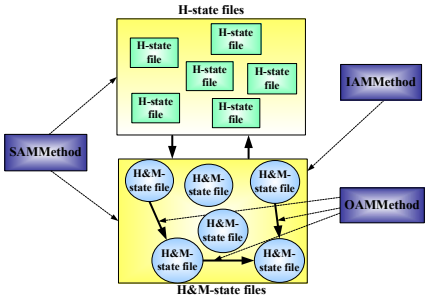

Figure 2. The framework of CMM

3 The monitor algorithms in CMM

3.1 SA monitor algorithm

The main function of this algorithm is to monitor whether H-State attributes of any file in an application are tampered. And at running time, it adopts Inverse Priority Multi-Terminals Distribute(IPMTD) monitor policy. According to running state of the application, the algorithm concludes two stages: pre-processing and run-time monitoring.

```
H-State Monitor Algorithm
----------
Step1 : Pre-processing
Begin
    Build the redundancy backup and store the FILE;
    While(hasNextIn(FILE))
    Begin
     MD5(filei);
     store(AttrOfFilei);
    End
End
Step2: Running （IPMTD Algorithm）
Begin
   Server：
      Build the FileAttrTable；
      //Check the File periodically;
      periodCheckThread（）；
      sendToClient（FileURL，FileName，Hashcode）；
   Client:
     listenToServer();
     getInfoFromServ();
     //if a file is changed, send its information to the
     //Determination mechanism
     IF(fileChanged())
       sendToDetermin(FileURL，FileName);
End
```

Figure 3. H-State monitor algorithm

3.2 IA monitor algorithm

Files in memory can be formally described as a quadruplet $R=(F,E,C,Event)$. $F=\{F_1,F_2,...,F_n\}$ is the set of all files, and $E=\{E_1,E_2,...,E_n\}$ is the set of exceptions which can cause failure of components, $C=\{T，Res\}$ is the set of determinant conditions of components' failure, including executing time T and resource occupancy rate Res, $Event=\{Event_1,Event_2...,Event_n\}$ is the set of monitored events, $F\times E\times C\rightarrow Event$.

IA monitor algorithm is shown in Figure 4.

```
Dynamic Inside Attribute Monitor Algorithm
----------
① Insert Probei into every component Fi;
② Probei{
   …
   try{…}
   //e∈Ei，Ei={exceptioni1,exceptioni2,…, exceptionin} is the
   //exception set of Fi
   catch(e){
     …
     new Eventi(e);
     sendToDetermin(Eventi); }
   }
③ //let the SysTime=k1*Tn, Tn is the normal runtime of Fi. SysTime
   //represents the  most tolerant time of the system. k1 is defined by
   //the developer.
  if(Te>SysTime) new Eventi(TimeLong);
   //let the UserTime=k2*Tn, UserTime represents the  most tolerant
   //time of the user. k2 is defined by the developer. k2> k1
  if(Te>UserTime) new Eventi(TimeOut);
  if(Eventi!=null)
    sendToDetermin(Eventi);
④ //Res is the RCR(Resource Consumption Rate) of Fi
   Res=getSystemResource();
   //Resn is RCR threshold of Fi which is measured during the normal
   //running time
   if(Res>Resn){
     new Eventi(ResourceOut);
     sendToDetermin(Eventi);}
⑤ go ahead while there's nothing wrong;
```

Figure 4. Dynamic IA monitor algorithm

3.3 OA monitor algorithm

```
Dynamic Outside Attribute Monitor Algorithm
---------------------------------------------------------------------
① //the independent node n represents the function
   //module in the system
   build the set N={n1,n2,...};
② for(i=0;i<N.lenth();i++)
    for(k=0;k<N.lenth();k++){
     if(hasMessages(ni,nk))
       Edgeik=true; }
③ //while Edgeik is true, the nodes ni andnk are adjacent
   build the set T={t1,t2,...};
   ti=getResbonseTime(adjacency(ni,nk));
④ //the smallest workflow unit wi is the process between
   //the adjacent nodes
   build the set W={w1,w2,…};
⑤ start to monitor wi when ni sends message;
⑥ UserTolerateTime(wi)=3*ti;
   if(OperTime(wi)==1.5*ti{
    new Eventi(FlowTimeOut);
    sendToDetermin(Eventi); }
⑦ end this monitor process;
```

Figure 5. Dynamic OA monitor algorithm

This algorithm is used to monitor communication between components in an application. If messages between components are able to be transmitted, the communication is regarded as normal regardless of the correctness of its business logic. The algorithm is used to monitor communication attributes between nodes in the work flow of an application while each component is regarded as a single node. The algorithm is shown in Figure 5.

4. Case study: the EShop

Our experiment environment concludes 20 host computers with the same configuration in the LAN of our lab: Intel Pentium4 3.0GHz CPU, 2GB memory, Window XP system, in which there are 1 DB server, 1 web server and 18 client terminals.

4.1 Target system

We select a typical teaching case EShop as target system for implementing CMM. EShop is a Java EE web application based on structure of JSP+AJAX+Strusts+ Hibernate. And it's a highly simulative web application which follows MVC model and adopts the popular open source ORM middleware Hibernate. EShop represents a typical web application, therefore it's appropriate to take EShop as the case to prove and improve practicality and generality of CMM.

4.2 Results and valuation

Performance cost of the proposed comprehensive monitor mechanism can be almost ignored because it only occupies very little memory and bandwidth. To verify the proposed CMM and algorithms, we injected faults into EShop and fault types, injected times, successful monitoring times, etc. are shown in Table 2. Seen from the table, 1400 times, 10 types of faults are injected. Thereinto, 1352 times are monitored, and 1291 times are recovered successfully.(successful recovery is the ones that users can't perceive) Among the three attributes' monitoring, SA's successful monitor rate is 100%, and the successful recovery rate is 92.5%; IA's successful monitor rate is 95% while the successful recovery rate is 91.3%, and OA's successful monitor rate is 95% with the successful recovery rate 95%. Average successful monitor rate of the injected faults is 96.4%, and average successful recovery rate is 92.1%. So we can see that average successful monitor rate is beyond 95%. Based on the monitor mechanism, average recovery rate of faults is above 90%.

Table 1 Partial experimental results

Monitor mechanism	Injected events	Injected times	Successful monitor times	Successful recovery times	Average recovery time（ms）
SA monitor	File lost	200	200	193	584
	Malicious attacks	200	200	179	580
IA monitor	declared exceptions	300	300	300	56
	OutOfMemoryError	100	91	84	264
	StackOverflowError	100	100	88	128
	IOException	100	100	97	78
	SQLException	100	89	80	785
	NullPointer	100	82	79	241

OA monitor	Response timeout	100	90	90	3870
	Misoperation of DB	100	100	100	4985

Seen from Table 2, average recovery time of the faults is 1157ms while average recovery time of file monitor(582ms) and average recovery time of runtime monitor(259ms) barely affect users. Since the recovery of misoperation of DB is manual behavior of managers, the average recovery time(4985ms) is acceptable.

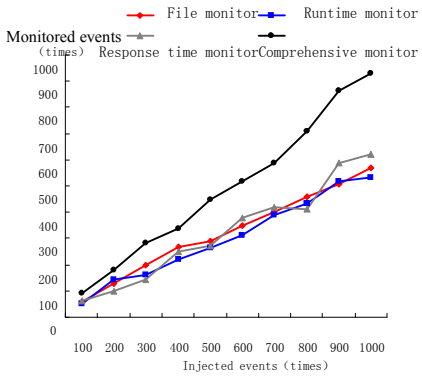

Figure 6 Monitor performance comparison curves

Events have transmissibility in an application, so many of them can be captured by different attributes monitors as the application runs. For example, the attack of worms injects codes which can consume resources into Actions. When resources are consumed too much, it will be captured by IA monitor. And response timeout could occur as the lack of resource. Then it will be captured by OA monitor. Besides, if this Action is not called for a long time, it will be written to external storage, in which case it will be captured by SA monitor as the file changes. The curves in Figure 6 are the performance comparison of single aspect monitor and comprehensive monitor. Seen from the figures, the missed rate of comprehensive monitor is obviously lower than that of single aspect monitor.

From Table 3, we can figure that the granularities of monitor and recovery in CMM are much finer than existing monitor mechanisms. This will lead to a fact that the following recovery means are more accurate. The recovery means of existing monitor are relatively coarser and they usually take longer time to recover. And compensation measures(like redundancy mirror technique) will cause an increase of TCO.

Table 2 Comparison to existing monitor mechanisms

Injected events	Monitor mechanism	Monitored probability	Recovery measure	Average recovery time（ms）
SA events	Exsiting means	low	Restart server	10486
	Proposed means	high	Hot swap component	582
IA events	Exsiting means	middle	Restart server or host	69526
	Proposed means	high	microreboot	259
OA events	Exsiting means	middle	Restart server or host	58431
	Proposed means	high	Microreboot, Undo	4427

5 Conclusion and future work

The main contribution of the paper is that we take general applications as target systems and build model for them, then we propose CMM towards self-healing which can improve recovery performance through reducing the grain size of monitor and recovery. From the perspective of information acquisition and realization, active and passive monitor, pre-processing and runtime monitor, static and dynamic attributes monitor means are integrated in our self-healing system. Different monitor means collaborate through monitor log and following recovery mechanisms.

The experimental results show that the proposed comprehensive multi-attributes monitor model can improve performance of self-healing by reducing the grain of monitor and recovery, and deduce performance cost of following recovery so that it has higher monitor performance and lower missed rate than single aspect monitors.

In the future, automatic process of each attribute monitor will become research emphasis and the monitored attributes and parameters will be extended to make the information obtained by CMM more complete.

References:

[1] Jeffrey O.Kephart, David M. Chess. The Vision of Autonomic Computing. Computer January2003, 41-50

[2] Lin Jing, etc. Method of data tamper detection by using improved MD5 algorithm. Computer Engineering and Applications'2008(33): 148-150

[3] Yang Feng, etc. Research of Reliability Maintenance and Adaptive Self-Recovery Mechanism in DHT Based Peer-to-Peer Super-Peer Networks. Computer Applications, 2004(5): 1-3

[4] Li Aiguo. Research about Distributed Software Fault Injection and Software Vulnerabilities Identifying, china doctoral dissertations full text database, 2008（12）

[5] Breitgand, David, Goldstein, etc. PANACEA-Towards a self-healing development framework. In proceeding of 10th IFIP/IEEE International Symposium on Integrated Network Management 2007, IM '07. 169-178

[6] Zhang Xin, etc. A Fault Monitoring Algorithm for Hierarchical Network. Journal of Electronics & Information Technology, 2007(4): 771-775

[7] Liu Donghong, etc. Monitoring Enabled Trusted Software Construction and Runtime Framework. National Software and Application Conference（NASAC2009）, 195-201. 2009

[8] Bao Jing, etc. A active and pas s ive mixture mechanism for self-recovery of P2P. Network and communication, 2008(24): 109-111

[9] George Candea, Mauricio Delgado, Michael Chen, Armando Fox. Automatic Failure-Path Inference: A Generic Introspection Technique for Internet Applications. In Proceedings of the the third IEEE Workshop on Internet Applications (WIAPP'03). 2003

[10] Jimeng Sun, Evan Hoke, John D. Strunk, Gregory R. Ganger, Christos Faloutsos.Intelligent System Monitoring on Large Clusters. Proceedings of the 3rd International Workshop on Data Management for Sensor Networks (DMSN'06), Seoul, South Korea, 2006.

[11] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey,S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. The VLDB Journal, 12(2):120–139, 2005.

[12] R. Buyya. PARMON：a portable and scalable monitoring system for clusters. Software - Practice and Experience，30(7): 723–739, 2000.

[13] Anderson E, Patterson D. Extensible, scalable monitoring for clusters of computers. Proceedings of the 11th Systems Administration Conference (LISA '04), San Diego, CA, October 2004.

[14] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide area cluster monitoring with ganglia. In CLUSTER, 2003.

[15] Jin Hai. Server monitor technique. http://media.ccidnet.com/media/ciw/1105/e1201.htm，2002

[16] HP OpenView. http://www.Managementsoftware.hp.com/index.html

[17] Wang Jilong，Wu Jianping. An Object-Oriented Model of Network Performance monitoring System. Journal of Software. 2001, 12(1): 18-25.

[18] Li Tao. An Immune Based Model for Network Monitoring.Chinese Journal of Computers. 2006,29(9): 1515-1522.

[19] Cha Li, etc. A LDAP Based Monitoring System for Grid. Journal of Computer Research and Development. 2002，39(8): 930-936.

[20] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS' 03), March 2003, 15-18.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (90718003), the Hi-Tech Research and Development Program of China (2007AA01Z401) and Fundamental Research Funds for the Central Universities：HEUCF100601