On Self-Aware Delay time based Service Request Optimization for Gateway Stability in Autonomic Self-Healing Systems

Junaid Ahsenali Chaudhry[†], Yonghwan Lee[‡], Seungkyu Park[†], Dugki Min[‡1]

[†]Graduate School of Information and Communication, Ajou University,
Woncheon-dong, Paldal-gu, Suwon, 443-749, Korea.
{junaid, sparky} @ ajou.ac.kr
[‡]School of Computer Science and Engineering, Konkuk University,
Hwayang-dong, Kwangjin-gu, Seoul, 133-701, Korea.
{yhlee, dkmin} @ konkuk.ac.kr

Abstract. The benefits of component-based service composition are immense however due to their exponential complexity; their real time implementation is a big challenge. In hybrid networks i.e. ubiquitous zone-based networks (u-Zone Networks), the high intensity of service requests can effect drastically on the performance stability of service gateways. In this paper, we present a self-aware service request optimization algorithm for autonomic self-healing systems. We propose when a service request is received and the system is found in the overload state where it cannot entertain more service requests, instead of imposing denial of service, the gateway would evaluate the workload of the worker thread at the gateway and reschedule the service request by deferring it. According to our simulation result, the proposed delay time algorithm, if implied, enable gateways with more stability in order to process high flux of service request loads even beyond the saturation point.

1 Introduction

The enormous management [1] cost is the trade-off of countless anticipated reimbursements by distributed networks i.e. u-Zone networks. The u-Zone networks carry the features of cluster-based Mobile Ad-hoc NETworks (MANETs) i.e. high heterogeneity, mobility, dynamic topologies, limited physical security, limited survivability, and low setup time [2] along with the support of high speed mesh backbones [3]. Autonomic Computing (AC) presents a novel solution for management costs in the form of self-management [4] but because of lack of standardization activities and demarcation of functional specifications, self-management is not a straight forward solution. Several network management solutions proposed in [5, 6, 7, 8] are confined strictly to their respective domains i.e. either mesh network or MANETs.

[1] Corresponding author: Dugki Min (dkmin@konkuk.ac.kr)

In u-Zone networks, due to such a wide variety of clients, it is not worthwhile to devise separate management solutions for each client. Rather we suggested that dividing the problem domain into sub domains [9]. The component-based solutions can be applied after dividing the problems into sub-domains. A component based self-healing solution is proposed in [10]. The dynamic service composition carries exponential complexity [11] so transaction thrashing can take place. The term thrashing generally describes "a phenomenon where an increase of the load results in decrease of throughput (or another-related performance measure)" [12]. So it is integral to provide a mechanism for preventing the thrashing in dynamic component integration systems. We use the delay time-based peak load control scheme in order to prevent the transaction trashing caused by enormous number of service request in u-Zone-based hybrid networks. This complexity may be the biggest hindrance in real-time implementation of autonomic solutions in real time environments.

Although there has not been much published work on service request management in self managing networks, we propose a time delay based peak load control mechanism for the self healing engine proposed in [11]. As the complexity of the solutions made by dynamic composition from components can be exponential [11], it is critical to provide incremental, low cost and time efficient solutions. The worker pattern [8], connecter-accepter model [16], reactor [15] and proactive [21] approaches are effective in combination but not cost effective in real life applications. The use of the Peak Load Control (PLC) mechanism manages the service requests at the gateway. Depending on the load on the host gateway, the service requests are evaluated and deferred to avoid Denial of Service (DoS). We assign time stamps to the service requests for "*fairness"* in their scheduling. We show the simulation result for proving the stability of performance. According to our experimental results, the proposed delay time algorithm can stably control the heavy overload after the saturation point and has significant effect on the controlling peak load.

In section 2 we discuss related work. The system architecture follows in section 3. In section 4 we discuss self-aware service request optimization mechanism. In section 5 we describe the simulation results along with their significance. In section 6 we conclude the paper and discuss future work.

2 Related Work

In this section we compare our research with related work. Since there has not been any research work published for controlling the service request load control in autonomic self healing networks, we can compare our scheme on the bases of autonomic self-management functions only. However the use of the self-aware PLC mechanism in self-healing networks remains uniqueness in the area.

The Robust Self-configuring Embedded Systems (*RoSES*) project [13] aims to target the management faults using self configuration. It uses graceful degradation as a means to achieve dependable systems. In [14] the authors propose that there are

certain faults that cannot be removed through configured of the system, which means that RoSES does not fulfill the definition of self management as proposed in [18]. The *HYWINMARC* [3] uses cluster heads to manage the clusters at local level but does not explain the criteria of their selection. The specifications of Mobile Code Execution Environment (*MCEE*) are absent. Moreover the use of intelligent agents can give similar results as discussed above in the case of [8, 15, 16, and 21]. To enforce management at local level, the participating nodes should have some local management entity. We compare the Autonomic Healing-based Self-management ENgine (*AHSEN*) with the other architectures. The comparison reveals that entity profiling, functional classification of self management entities at implementation level, and assurance of the functional compliance is not provided in the schemes proposed. Moreover the self monitoring at node's local level courtesy *NFM* not only gives a node its self awareness but also uses the shared medium to the minimum. In very dynamic hybrid networks these functionalities go a long way in improving the performance of the self management system.

3 System Architecture

In hybrid wireless networks, there are many different kinds of devices attached with the network. They vary from each other in their power, performance etc. One of the characteristics not present in the related literature is the separate classification of the client and the gateway architectures. Figure 1 shows the client and gateway self-management software architectures.

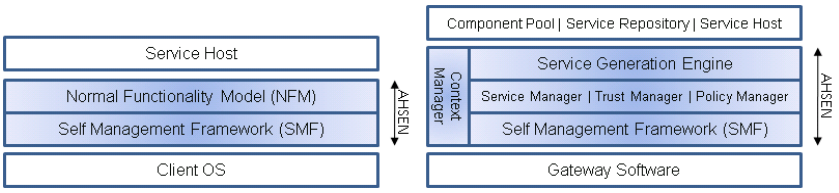


Fig. 1. The AHSEN architectures of client (a) and gateway (b)

The *Normal Functionality Model (NFM)* is a device dependent ontology that is downloaded, along with *SMF*, on the device at network configuration level. It provides a mobile user with an initial default profile at gateway level and device level functionality control at user level. The client *SMF* constantly *tracks* the user activities and sends them to the *SMF* at the gateway. The *SMF* at gateway directs the *track* requests to the context manager which updates the related profile of the user. The changes in service pool, trust manager, and policy manger are reported to the *Context Manager (CM)*. The *CM* consists of a *Lightweight Directory Access Protocol (LDAP)* directory that saves its sessions at predetermined time intervals in the gateway directory. The *Policy Manager (PM)* and *Service Manager (SM)* follow the same

registry based approach to enlist their resources. The presence of *NFM* provides decision based reporting unlike the ever-present *SNMP*. The Trust Manager uses the reputation-based trust management scheme in public key certificates [10]. The trust factor is typically decided on trustee's reputation to mitigate risks involved in interaction with unknown and potentially malicious users. We desist providing more details on AHSEN architecture here. For a more detailed description of the architecture shown in figure 1, please refer to [10].

In [3] the authors classify self management into individual functions and react to the anomaly detected through *SNMP* messages. The clear demarcation of *self-** functions is very difficult due to the original mapping of faults onto management functions is not defined. Considering the overlapping nature of faults, it is more fertile to target the problems, which can be either atomic or complex, with atomic, vaguely categorized solutions that either combine at run time to make complex 'healing policies' or work independently in the shape of components [12]. The service generation engine is a rule-based engine embedded into AHSEN for template based service generation from components, the details of which are not within the scope of this paper.

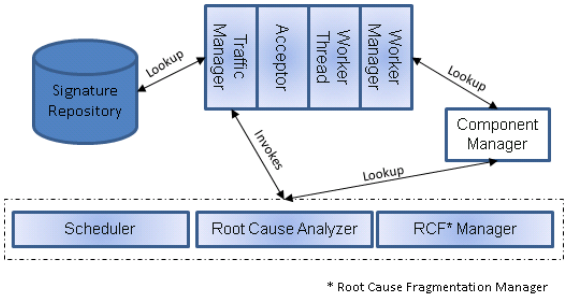


Fig. 2. Architecture of Self Management Framework (SMF)

The Root Cause Analyzer is the core component of the problem detection phase of healing. The State Transition Analysis based approaches [21] might not be appropriate as *Hidden Markov Models (HMMs)* takes long training time along with exhaustive system resources utilization. The profile based Root Cause Detection might not be appropriate mainly because of the vast domain of errors expected. Considering this situation, we use the meta-data obtained from *NFM* to trigger *Finite State Automata (FSA)* series present at the Root Cause Analyzer. In the future we plan to modify *State Transition Analysis Tool* [21] in lines of on fault analysis domains. After analyzing the root cause results from the *RCA*, the *RCF* manager in cooperation with the Signature Repository and Scheduler search for the already developed solutions else it arranges a time slot based scheduler for plug-ins. The traffic manager directs the traffic towards different parts of *AHSEN*.

4 Self-Aware Service Request Optimization

The *Traffic Manager* receives Simple Object Access Protocol (*SOAP)* requests from many devices within a cluster and redirects them to all the other internal parts of *SMF*. Figure 3 shows the structure of delay time-based peak load control. The *Acceptor* thread of the *Traffic Manager* receives a *SOAP* request (service request) and then puts it into the *Wait Queue*. The *Wait Queue* contains the latest context of the gateway load. If the gateway is in saturated state, the service request is handled by the self-aware sub module. The figure 4 is the pseudo code of self-aware sub-module in *WorkerManager's* Delay Time Algorithm. Let a service request (SR_1) arrives at the gateway. At first the SR_1 is checked if it contains the *comebacktime* stamp (for fair scheduling). If the *comebacktime* is 'fair' (that is the service request is returned after the instructed time), it is forwarded to the *Wait Queue* else it is accessed against the work load of the *Worker Manager*. The *Acceptor* is updated about the latest status of the Worker Manager. The Acceptor evaluates the intensity of current workload (how long it will take to free resources) and adds *buff* (buffer is the time to give some extra room to gateway) to the time. The aggregate time is assigned to SR_1 and the service request is discarded.

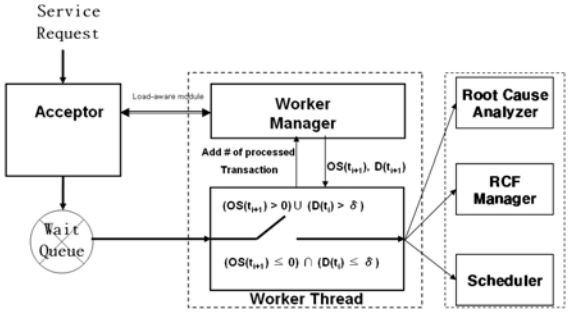


Fig. 3. The Structure of Delay Time-Based Peak Load Control in AHSEN

When the system is ready to accept the service request, the *Traffic Manager* get a *Worker Thread* from a thread pool and run it. The *Worker Thread* gets the delay time and the over speed from the *WorkerManager*. The admission to other internal parts SMF is controlled by the *Worker Thread* that accepts the arriving requests only if the over speed $OS(t_{i+1})$ at the time t_{i+1} is below zero and the delay time $D(t_i)$ at the time t_i is below the baseline delay δ. Otherwise the requests have to sleep for the delay time calculated by the *WorkerManager*. After the *Worker Thread* sleeps for the delay time, the *Worker Thread* redirects the requests to the Root Cause Analyzer, the *RCF* Manager, and the Scheduler. Finally, the *Worker Thread* adds the number of processed transaction after finishing the related transaction.

The figure 4 is the pseudo code for the *WorkerManager*'s delay time algorithm. After sleeping during interval time, the *WorkerManager* gets the number of transactions processed by all *Worker Threads* and the maximum transaction

processing speed configured by a system administrator. And then, the *WorkerManager* calculates the TPMS (Transaction per Milliseconds) by dividing the number of transactions by the maximum transaction processing speed and calculate the over speed $OS(t_{i+1})$ that means the difference of performance throughput at the time t_{i+1} between the TPMS and the maximum transaction processing speed during the configured interval time. If the value of the over speed is greater than zero, the system is considered as an overload state. Accordingly, it is necessary to control the overload state. On the contrary, if the value of the over speed is zero or less than zero, it is not necessary to control the transaction processing speed. For controlling the overload state, this paper uses the delay time algorithm of the *WorkerManager*. The Figure 5 describes the formulas for calculating the delay time.

```
0-  Let a service request SR1 arrives at Acceptor
1-  Check SRi.comebacktime
2-  If (SRi.comebacktime='fair') // check the virtual queue
        a.  Wait Queue ← Send SR1
3-  Acceptor ← Send current_context(worker_Manager_Status_Update)
4-  If (current_context!='overloaded')
        a.  Wait queue ← Send SR1
5-  else
        a.  While (current_context='overlaoded')
                i.  delaytime= Calculate
                    (intensity_of(current_context)+buff
               ii.  Set SR1.comebacktime ← delaytime
              iii.  Dismount SR1
6-  While run_flag equals "true" do
7-  get interval time for checking load
8-  sleep during the interval time
9-  get the number of transactions processed during the interval time
10- get the configured maximum speed
11- TPMS := number of transactions / interval time
12- over speed := TPMS – the configured maximum speed
13- If over speed >0 then
        a.  get the previous delay time
        b.  if previous delay time = 0
                i.  previous delay time := 1
        c.  get the number of active worker thread
        d.  new delay time:= over speed / number of active worker *
            previous delay time
14- else
        a.  get current delay
        b.  if current delay > δ
                i.  new delay time := current delay * β
        c.  else
                i.  new delay time := 0
        d.  end if
15-  end if
16- end while
```

Fig. 4. The Pseudo code for Self-Aware module and WorkerManager's Delay Time Algorithm

If the over speed $OS(t_{i+1})$ is greater than zero, the first formula of the Figure 5 is used for getting a new delay time $D(t_{i+1})$ at the time t_{i+1}. The $N(t_{i+1})$ means the number

of active *Worker Threads* at the time t_{i+1} and $D(t_i)$ means the delay time at the time t_i. If the $D(t_i)$ is zero, $D(t_i)$ must be set one. If the $OS(t_{i+1})$ is below zero and the delay time $D(t_i)$ at the time t_i is greater than the baseline delay δ, The $D(t_{i+1})$ is calculated by applying the second formula of the Figure 5. On the contrary, if the $D(t_i)$ is below the baseline delay, $D(t_{i+1})$ is directly set zero. In other word, because the state of system is under load, the delay time at the time t_{i+1} is not necessary. Accordingly, the *Worker Thread* can have admission to other internal parts *SMF*. The baseline delay is used for preventing repetitive generation of the over speed generated by suddenly dropping the next delay time in previous heavy load state. When the system state is continuously in state of heavy load for a short period of time, it tends to regenerate the over speed to suddenly increment the delay time at the time t_i and then suddenly decrement the delay time zero at the time t_{i+1}. In other words, the baseline delay decides whether next delay time is directly set zero or not.

The β percent of the second formula of the Figure 5 decides the slope of a downward curve. However, if the delay time at the time t_i is lower than the baseline delay. The new delay time at the time t_{i+1} is set zero. Accordingly, when a system state becomes the heavy overload at the time t_i, the gradual decrement by β percent prevents the generation of repetitive over speed caused by abrupt decrement of the next delay time.

$$D(t_{i+1}) := \begin{cases} \dfrac{OS(t_{i+1}) * D(t_i)}{N(t_{i+1})}, if(OS(t_{i+1})) > 0 \\ D(t_i) * \beta, if((OS(t_{i+1})) \le 0 \cap (D(t_i) > \delta)), \\ 0, if((OS(t_{i+1})) \le 0 \cap (D(t_i) \le \delta)) \end{cases}$$

Fig. 5. A Mathematical Model for Delay Time Calculation

Once the service request is received by the worker thread the analysis of the cause of anomaly starts. As proposed in [18] the faults can be single root cause based or multiple root cause based. We consider this scenario and classify a *Root Cause Analyzer* that checks the root failure cause through the algorithms proposed in [19]. After identifying the root causes, the *Root Cause Fragmentation Manager (REF Manager)* looks up for the candidate plug-ins as solution. The *RFC manager* also delegates the candidate plug-ins as possible replacement of the most appropriate. The scheduler schedules the service delivery mechanism as proposed in [20]. The processed fault signatures are stored in signature repository for future utilization.

5 Simulation Results

In order to prove performance stability of the self-aware PLC-based autonomic self-healing system, we simulated the self-aware delay time algorithm of the

WorkerManager. As for load generation, the *LoadRunner 8.0* tool is employed. The delay time and over speed are used as a metric for simulation analysis. The *maximum speed*, δ and β for delay time algorithm are configured 388, 100ms, and 0.75 respectively. Figure 6 shows the result of simulation for describing the relationship between the over-speed and the delay time after the saturation point.

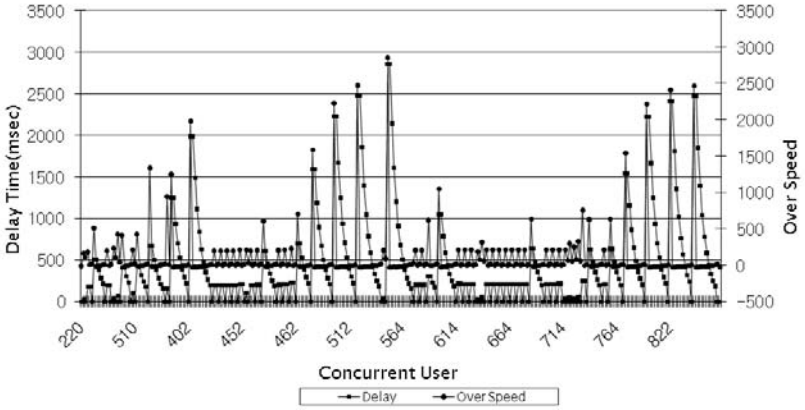
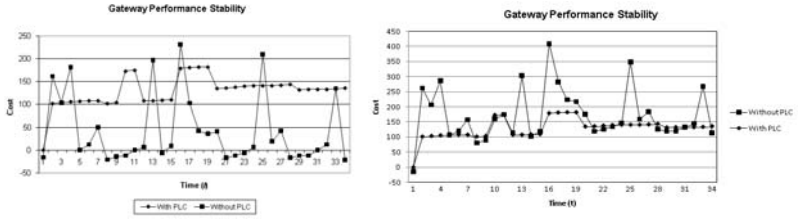


Fig. 6. The Simulation Results Proving the Effect of Delay Time Algorithm

This experimental results prove that the proposed delay time algorithm of the *WorkerManager* has an effect on controlling the over-speed. As the number of concurrent users is more than 220 users, the over-speed frequently takes place. Whenever the over-speed happens, each *Worker Thread* sleeps for the delay time calculated by the *WorkerManager*. As the higher over-speed takes place, each *Worker Thread* sleeps for the more time so that the over speed steeply goes down. Although the over speed steeply goes down, the delay time does not steeply goes down due to the baseline delay value *δ*. As the baseline delay value is set 100 ms in this experiment, the delay time gradually goes down until the 100 ms. As soon as the delay time passes 100 ms, the next delay time is directly set zero. The result of simulation in figure 7.a shows that the over-speed does not happen until zero delay time due to the slope of a downward curve. However, As soon as the delay time passes zero, the over speed again occurs and the next delay time controls the over speed.

Although the heavy request congestion happens in a *Traffic Manager* of the gateway, the delay time-based *PLC* mechanism can prevent the thrashing state in overload phase and help the *Traffic Manager* to execute stably the management service requests. The figure 7.a shows that the gateway with PLC scheme is more stable than the one without PLC mechanism. The standard deviation at the gateway without PLC is more than 58.23 whereas the deviation in performance cost at the gateway with PLC mechanism is 24.02 which prove the argument posted in the previous section that PLC mechanism provides stability to gateways in u-zone based networks. The figure 8.b shows that applying self-aware sub module to the PLC mechanism gives stable performance than applying PLC algorithm only. The stability

in the cost function with time shows that the cost is predictable over time scale. Although the cost of applying PLC mechanism with self-aware module is more than without it, the self-aware PLC gives more stability hence is more suitable in unpredictable, dynamic, and highly heterogeneous u-Zone Networks.



(a) (b)

Fig. 7. The Simulation Results for Gateway Performance Stability using WorkerManager's Delay Time Algorithm

6 Concluding Remarks and Future Work

In this paper, we propose a self-aware delay time based service request optimization algorithm for gateway stability in ubiquitous networks. We apply the scheme in autonomic self-healing based systems. As there is large variety of clients trying to access the same set of services it is highly probable that a service that works optimally for a certain type of client may prove terminal for another type of client. So we propose to decrease the granularity and name them *components* and join them dynamically. We identify that because of exponential complexity among dynamic service composition systems, their real time implementation is not easy. For this reason, we propose a self-aware delay time based algorithm that gives more stability to the gateway than some of the solutions proposed.

In future we aim to test this scheme in more complex situations and at mutually dependent services. We aim to improve the root cause analysis algorithms so that the exact situation at the client end could be sorted out. The performance of self-aware enabled PLC algorithm is yet to be tested in the presence of multiple users.

References

1. Firetide™ Inc. www.firedide.com. Last accesses: 2007-01-01.
2. Doufexi, A. Tameh, E. Nix, A. Armour, S. Molina, A. "Hotspot wireless LANs to enhance the performance of 3G and beyond cellular networks", Communications Magazine, IEEE, Publication Date: July 2003, Volume: 41, Issue: 7,On page(s): 58- 65.

3. Minseok Oh. Network management agent allocation scheme in mesh networks Communications Letters, IEEE Volume 7, Issue 12, Dec 2003 Page(s):601 – 603.
4. Cybenko, G.; Berk, V.H.; Gregorio-De Souza, I.D.; Behre, C., "Practical Autonomic Computing," Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International , vol.1, no.pp.3-14, Sept. 2006
5. Shafique Ahmad Chaudhry, Ali Hammad Akbar, Ki-Hyung Kim, Suk-Kyo Hong, Won-Sik Yoon," HYWINMARC: An Autonomic Management Architecture for Hybrid Wireless Networks" Network Centric Ubiquitous Systems (NCUS 2006).
6. Burke Richard, 2004, "Network Management. Concepts and Practice: A Hands-on Approach", Pearson Education, Inc.
7. Kishi Y. Tabata, K.; Kitahara, T.; Imagawa, Y.; Idoue, A.; Nomoto, S.; Implementation of the integrated network and link control functions for multi-hop mesh networks in broadband fixed wireless access systems Radio and Wireless Conference, 2004 IEEE 19-22 Sept. 2004 Page(s):43 - 46
8. S. Yong-Lin, G. DeYuan, P. Jin, S. PuBing, A mobile agent and policy-based network management architecture, Proceedings, Fifth International Conference on Computational Intelligence and Multimedia Applications ICCIMA 2003, 27-30 Sept. 2003, Page(s):177-181.
9. Junaid Ahsenali Chaudhry, Seungkyu Park, "A Novel Autonomic Rapid Application Composition Scheme for Ubiquitous Systems", The 3rd International Conference on Autonomic and Trusted Computing (ATC-06), 2006
10. Junaid Ahsenali Chaudhry, and Seungkyu Park, "Using Artificial Immune Systems for Self Healing in Hybrid Networks", in Encyclopedia of Multimedia Technology and Networking, Published by Idea Group Inc., 2006. [To appear]
11. Turing, Alan M., On Computable Numbers, with an Application to the Entscheidungs Problem. Proceedings of the London Mathematical Society, 2 (42):230-265, 1936.
12. P. J. Denning: Thrashing: Its Causes and Prevention. Proc. AFlPS FJCC 33, 1968, pp, 915-922
13. Shelton, C. & Koopman, P., "Improving System Dependability with Alternative Functionality," DSN04, June 2004.
14. Morikawa, H. (2004). The design and implementation of context-aware services. Proceedings of IEEE saint-w 2004, 293 – 298.
15. D. C. Schmidt, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching," in Pattern Languages of Program Design (J. O. Coplien and D. C. Schmidt, eds.), pp. 529–545, Reading, MA: Addison-Wesley, 1995
16. D. C. Schmidt, "Acceptor and Connector: Design Patterns for Initializing Communication Services," in Pattern Languages of Program Design (R. Martin, F. Buschmann, and D. Riehle, eds.), Reading, MA: Addison-Wesley, 1997.
18. Wolfgang Trumler, Jan Petzold, Faruk Bagci, Theo Ungerer, AMUN – Autonomic Middleware for Ubiquitious eNvironments Applied to the Smart Doorplate Project, International Conference on Autonomic Computing (ICAC-04), New York, NY, May 17-18, 2004.
19. Gao, J.; Kar, G.; Kermani, P.; Approaches to building self healing systems using dependency analysis, Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP Volume 1, 19-23 April 2004 Page(s):119 - 132 Vol.1
20. Junaid Chaudhry, and Seungkyu Park, "On Seamless Service Delivery", The 2nd International Conference on Natural Computation (ICNC'06) and the 3rd International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'06) 2006.
21. J. Hu, I. Pyarali, and D. C. Schmidt, "Applying the Proactor Pattern to High-Performance Web Servers," in Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems, IASTED, Oct. 1998.