DOI : 10.3745/JIPS.2009.5.2.069

Autonomic Self Healing-Based Load Assessment for Load Division in OKKAM Backbone Cluster

Junaid Ahsenali Chaudhry*

Abstract: Self healing systems are considered as cognation-enabled sub form of fault tolerance system. But our experiments that we report in this paper show that self healing systems can be used for performance optimization, configuration management, access control management and bunch of other functions. The exponential complexity that results from interaction between autonomic systems and users (software and human users) has hindered the deployment and user of intelligent systems for a while now. We show that if that exceptional complexity is converted into self-growing knowledge (policies in our case), can make up for initial development cost of building an intelligent system. In this paper, we report the application of AHSEN (Autonomic Healing-based Self management Engine) to in OKKAM Project infrastructure backbone cluster that mimics the web service based architecture of u-Zone gateway infrastructure. The 'blind' load division on per-request bases is not optimal for distributed and performance hungry infrastructure such as OKKAM. The approach adopted assesses the active threads on the virtual machine and does resource estimates for active processes. The availability of a certain server is represented through worker modules at load server. Our simulation results on the OKKAM infrastructure show that the self healing significantly improves the performance and clearly demarcates the logical ambiguities in contemporary designs of self healing infrastructures proposed for large scale computing infrastructures.

Keywords: *Self Healing Systems, Load Estimation and Balancing, OKKAM, Entity Naming System*

1. Introduction

As the complexity and size of networks increase so does the costs of network management [1]. The preemptive measures have done little to cut down on network management cost. Hybrid networks cater with high levels of Quality of Service (QoS), scalability, and dynamic service delivery requirements. The amplified utilization of hybrid networks i.e. ubiquitous-Zone based (u-Zone) networks has raised the importance of human resources, down-time, and user training costs. The u-Zone networks are the fusion of the cluster of hybrid Mobile Ad-hoc NETworks (MANETs) and high speed mesh network backbones. They provide robust wireless connectivity to heterogeneous wireless devices and take less setup time. The clusters of hybrid networks feature heterogeneity, mobility, dynamic topologies, limited physical security, and limited survivability [2] and the mesh networks provide the high speed feedback to the connected clusters.

Autonomic Computing provides a cheaper solution for robust network management in u-Zone networks in the form of self-management. Self Management is a tool through which performance of the computer systems can be optimized without human user intervention. In [14] Turing et. al. suggests that autonomic systems have exponential complexity which can hamper the appropriate problem marking and also raises the software cost. So it is critical to provide incremental, low cost and time efficient solution along with minimize the maintenance cost of the software.

An Entity Name System (ENS) – as it is currently under development in the EUfunded project OKKAM – for systematically supporting the re-use of entity identifiers. The main purpose of the ENS is to provide unique and uniform names for entities for the use in information collections, so that the same name is used for an entity, even when it is referenced in different contexts.

At this stage the ENS V2 architecture is envisaged to consist of five main elements:

- an OKKAMcore Cluster, i.e. a set of computers running OKKAMcore, together with a standard Load Balancer that interfaces with the outside world.
- Lucene/Hadoop Cluster for distributed and load-balanced index and entity profiles management. This may be implemented using Solr/Hbase if experimentation proves that it is better for future scalability and reliability to use SolrBroker in the OKKAMstore.
- Security Server

Manuscript received 16 January, 2009; revised 16 February, 2009; accepted 25 February, 2009.

Corresponding Author: Junaid Ahsenali Chaudhry
* University of Trento, Italy (chaudhry@disi.unitn.it)

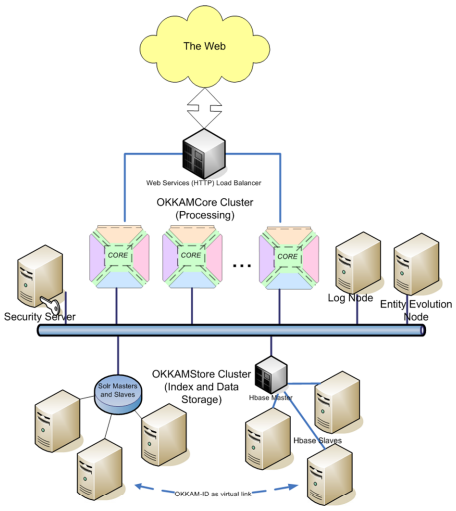

Fig. 1. Concrete Architecture of the Public OKKAM ENS

- storage and processing node suitable for analysis of log data
- storage and processing node suitable for management of entity evolution

Within this architecture, the instances of OKKAMstore which are deployed as part of OKKAMcore will take care of interaction with the Lucene/Hadoop (or Solr/HBase) clusters transparently (i.e. any additionally deployed instance of OKKAMcore will address the same clusters automatically through the OKKAMstore component).

The focus of this scaled up and distributed ENS V2 will be much more on operational performance metrics, and to quantify the quality of service it will provide to its users, particularly those of the three OKKAM entity-centric applications in an authoring environment, organizational knowledge management and entity-centric search engine. In its use, it will also enable the ENS service performance and quality expectations and needs of those users to be also quantified. These will then inform the specification, implementation, practical operation and roadmap to the final large-scale ENS V3 to emerge from this project.

The authors in [17] target the self management in hybrid environment through 'divide and conquer' approach by using component-based programming. They propose to rapidly divide the problem into sub domain and each domain is than assigned 'sub solutions'. The amalgamation of all 'sub solutions' gives the final management solution to the client.

Several network management solutions proposed in [4-7] are confined strictly to their respective domains i.e. either mesh network or MANETs. A self management architecture is proposed in [30] for u-zone networks that uses component-based plug-ins as self healing policies. Different executable components are integrated together to form a healing policy. A case study is presented in this paper as an application of the AHSEN architecture. We use the delay time-based peak load control scheme in order to preventing the transaction trashing caused by enormous number of service request in u-Zone-based hybrid networks. The 'blind' load division on per-request bases is not optimal for distributed and performance hungry infrastructure such as OKKAM. The approach adopted assesses the active threads on the virtual machine and does resource estimates for active processes. The availability of a certain server is represented through worker modules at load server. The worker pattern [8], connecter-accepter model [16], reactive [15] and proactive [26] approaches are effective in combination but not cost effective in real life applications. Use of Peak Load Control (PLC) mechanism manages the service requests at gateway. Depending upon the load on the host gateway, the service requests are routed to the peer gateways to eliminate the inconsistency and redundancy at service level. We show the simulation result for proving the stability of performance. According to our experiment result, 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 compare our scheme with some of the contemporary solutions proposed. The proposed scheme follows in section 3. The results are reported in section 4 and we conclude this paper in section 5.

2. Related Work

In this section we compare our research with the related work. The Robust Self-configuring Embedded Systems (RoSES) project [13] aims to target the management faults using self configuration. It uses graceful degradation as means to achieve dependable systems. In [14] the authors propose that there are certain faults that can not 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 gives similar results as discussed above in the case of [16] and [17, 29]. To enforce the management at local level, the participating nodes should have some management liberty. However HYWINMARC fails to answer the questions rose in the previous section.

Service Synthesizer on the Net (STONE) project [14] explores new possibilities for users to accomplish their tasks seamlessly and ubiquitously. The project focuses on

the development of context-aware services in which applications are able to change their functionality depending on the dynamically changing user context. In STONE project the service discovery is considered as one of the general resources on internet.

We compare the architectures discussed in this section in a table to observe their efficacy. In table 1 we compare the AHSEN with the other architectures. The comparison reveals that the 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 uses the shared medium to the minimum also. In very dynamic hybrid networks these functionalities go a long way in improving the performance of the self management system.

3. Proposed Architecture

The Traffic Manager receives SOAP requests from many devices within a cluster and redirects them to all the other internal parts of SMF. Fig. 5 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 Fig. 6 is the pseudo code of self-aware sub-module in WorkerManager's Delay Time Algorithm. Let a service request (SR1) arrives at the gateway. At first the SR1 is checked if it contains the comebacktime stamp (for fair scheduling). If the come-backtime 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 SR1 and the service request is discarded.

When the system is ready to accept the service request, the Traffic Manager gets 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(ti+1) at the time ti+1 is below zero and the delay time D(ti) at the time ti is below the baseline delay δ. Otherwise the requests have to sleep for the delay time

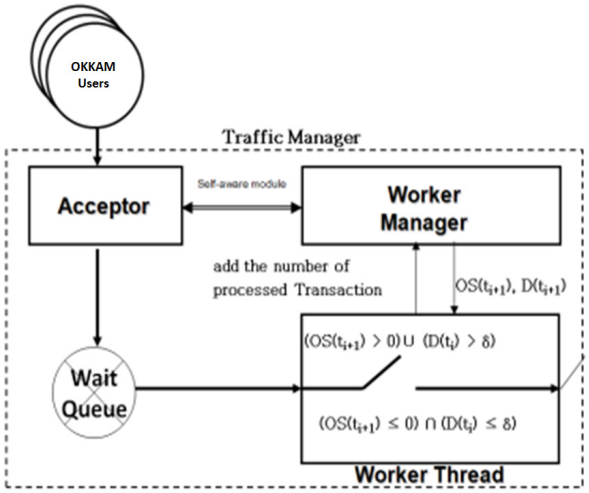

Fig. 2. Structure of Delay-time based Peak Load Control Scheme

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. 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(ti+1) that means the difference of performance throughput at the time ti+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 Fig. 7 describes the formulas for calculating the delay time. If the over speed OS(ti+1) is greater than zero, the first formula of the Fig. 6 is used for getting a new delay time D(ti+1) at the time ti+1. The N(ti+1) means the number of active Worker Threads at the time ti+1 and D(ti) means the delay time at the time ti. If the D(ti) is zero, D(ti) must be set one. If the OS(ti+1) is below zero and the delay time D(ti) at the time ti is greater than the baseline delay δ. On the contrary, if the D(ti) is below the baseline delay, D(ti+1) is directly set zero. In other word, because the state of system is under load, the delay time at the time ti+1 is not necessary.

Accordingly, the Worker Thread can have admission to

```
0-  Let a OKKAM service request SR1 arrives at
    Acceptor
1-  Check SR1.comebacktime
2-  If (SR1.comebacktime='fair') // check the
    virtual queue
        a.  Wait Queue ← Send SR1
3-  Acceptor ← Send
    current_context(worker_Manager_Status_Updat
    e)
4-  If (current_context!='overloaded')
        a.  Wait queue ← Send SR1
5-  else
        a.  While
            (current_context='overlaoded')
                i.  delaytime= Calculate
                    (intensity_of(current_cont
                    ext)+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. 3. The Pseudo Code for Self-Aware Module and WorkerManager's delay time Algorithm

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

Fig. 4. A Mathematical Model for Delay time Calculation.

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 ti and then suddenly decrement the delay time zero at the time ti+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 Fig. 7 decides the slope of a downward curve. However, if the delay time at the time ti is lower than the baseline delay. The new delay time at the time ti+1 is set zero. Accordingly, when a system state becomes the heavy overload at the time ti, the gradual decrement by β percent prevents the generation of repetitive over speed caused by abrupt decrement of the next delay time.

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. Let, N is concurrent service requests at the server at full time. This means that as soon as one thread finishes its execution, a new one will take its place. This assumption is made in order to insure that we analyze the worst case scenario of performance time with N service requests in execution queue. This means that the execution of a service request is done from its first until its last quantum (subparts of a service request i.e. analyze, evaluate, categorize etc) in the presence of other N-1 service requests.

- k→ index variable spanning the service requests: $1<=k<=N$
- S_j→ CPU quantum length for server j (isolated represents that value for a specific isolated server)
- Ik_j→the number of cycles the k^{th} service request needs to complete on server j.

$i_{k\_isolated}$ represents that value for a specific isolated server

As soon as the execution queue is in a stable state, the time needed for the k^{th} service request to complete in the presence of other N-1 service requests is

$$t_k = i_{kj} * s_j * N \tag{1}$$

The $i_{kj} * s_j$ this product represents in fact the execution time of the kth service request in isolation conditions (executed alone, without any other concurrent thread).

This product $t_{k\_isolated} = i_{k\_isolated} * s_{isolated}$ is evaluated on an out of core server and is used as the base value for the load prediction. These two formulas are rather trivial and are standard results of queuing system. They

mean that the prediction time for the execution depends on the total amount of connected service requests on the server.

Now a transaction is completed client request, and considering that the kth client permanently issues the same request to the server, then the number of transactions (Tx) that may be completed for k clients in interval T is

$$Tx_k = \frac{T}{i_{kj} * s_j * N} \tag{2}$$

from 1,

$$Tx_k = \Delta * \frac{T}{t_{k\_isolated} * N} \tag{3}$$

where Δ is coefficient of CPU utilization. We can write 3 as,

$$Tx\text{cpu}_k = \Delta * \frac{T}{t_{k\_isolated} * N} \tag{4}$$

From equation 4, it means that the execution time for a transaction depends upon the number of service requests in active queue. So we can calculate the estimated time a CPU needs in order to get free from the requests in active queue.

Let, there are N number of service requests present in the active queue. In time tk1, the service request sk1 is being executed, the KN-1 service requests will reside in the memory.

Msk→ memory size kth service request. Msk >=1

Bk→ branch statements in Msk in kth service request. Bk >=0

Ttpbk→ the Time needed per transaction.

$$Txmem_k = \emptyset * \left(\frac{M_{s_k} * B_k}{T_{tpb_k}} * N\right) \tag{5}$$

where $\emptyset$ is coefficient of CPU utilization.

Let, u(t) denote load of service request. We can normalize the service request as

$$y(t) = \frac{u(t) - u_{min}(t)}{u_{max}(t)} \tag{6}$$

Where $u_{min}(t)$ and $u_{max}(t)$ denotes the minimum and maximum load of the service request. According to equation 6, different service request traces and be compared with each other, while the impact of their internal analysis is eliminated. If we define T_k as the kth threshold for $k = 0,1,2,\dots,K$ then a function $y_{T_k}(t)$ is defined by

$$y_{T_k}(t) = \begin{cases} 1, & y(t) \geq T_k \\ 0, & \text{else} \end{cases} \tag{7}$$

Generally $T_k = k * \frac{1}{K}$. Assuming $y_{T_k}(t) = 1$. From 6 and 7 we can say

$$T_k \leq \frac{u(t) - u_{min}(t)}{u_{max}(t)} \tag{8}$$

Or we can say,

$$T_{nwbuff_k} = \delta * \left\{\frac{u(t) - u_{min}(t)}{u_{max}(t)}\right\} \tag{9}$$

where δ is coefficient of CPU utilization.

So combining 4, 5, 9 we get,

$$T_{comback} = Txcpu_k + Txmem_k + T_{nwbuff_k} \tag{10}$$

$$T_{comback} = \left\{\Delta * \frac{T}{t_{k\_isolated} * N}\right\} + \left\{\emptyset * \left(\frac{M_{s_k} * B_k}{T_{tpb_k}} * N\right)\right\} + \left\{\delta * \left(\frac{u(t) - u_{min}(t)}{u_{max}(t)}\right)\right\} \tag{11}$$

Now we know that the service request has exponential distribution and the arrival rate has poison distribution. So we can say that the RTD (Round Trip Delay)

$$RTD = T_{comeback} + queuing\ delay + \tau_f + \tau_e \tag{12}$$

Where τ_f is the propagation delay from server to the bottleneck link buffer that is the gateway buffer, and τ_e is the propagation delay over the return path from the bottleneck link buffer to the client (the service request generator).

$$RTD = \left\{\Delta * \frac{T}{t_{k\_isolated} * N}\right\} + \left\{\emptyset * \left(\frac{M_{s_k} * B_k}{T_{tpb_k}} * N\right)\right\} + \left\{\delta * \left(\frac{u(t) - u_{min}(t)}{u_{max}(t)}\right)\right\} + queuingdelay + \tau_f + \tau_e \tag{13}$$

And efficiency is calculated as

$$E(S * N) = \frac{N * T_s}{S * 1 * T(S * 1)} \approx \frac{N * T_s}{(S) S * T} \tag{14}$$

In this expression, the range for S is $\left[1 \dots \frac{N}{n}\right]$ and the range for N is $[1..k * \log_n N]$ which is less then $[2\log_n N]^n$ for HYWINMARC [3] based, and $\left[2\log_n \frac{N}{2} + N^{n^2}\right]^n$ for RoSeS [13] based solutions.

4. 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. Fig. 8 shows the result of simulation for describing the relationship between the over-speed and the delay time after the saturation point.

These 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 Fig. 9.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.

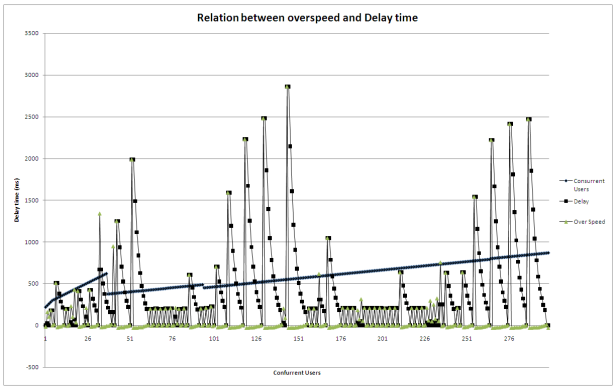

Fig. 5. Simulation Results showing the effects on performance.

5. Conclusion

In this paper we identify the role of self healing which is mostly misunderstood among the modern day systems. This misconception creates many logical problems especially in fault mapping, functional classification and categorization. Self healing systems are considered as cognation-enabled sub form of fault tolerance system. But our experiments that we report in this paper show that self healing systems can be used for performance optimization, configuration management, access control management and bunch of other functions. The exponential complexity that results from interaction between autonomic systems and users (software and human users) has hindered the deployment and user of intelligent systems for a while now. We show that if that exceptional complexity is converted into self-growing knowledge (policies in our case), can make up for initial development cost of building an intelligent system. In this paper, we report the application of AHSEN (Autonomic Healing-based Self management Engine) to in OKKAM Project infrastructure backbone cluster that mimics the web service based architecture of u-Zone gateway infrastructure. The 'blind' load division on per-request bases is not optimal for distributed and performance hungry infrastructure such as OKKAM. The approach adopted assesses the active threads on the virtual machine and does resource estimates for active processes. The availability of a certain server is represented through worker modules at load server. Our simulation results on the OKKAM infrastructure show that the self healing significantly improves the performance and clearly demarcates the logical ambiguities in contemporary designs of self healing infrastructures proposed for large scale computing infrastructures.

References

[1] Firetide www.firedide.com.

[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, Vol.41, Issue7,On pp.58-65.

[3] 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).

[4] Burke Richard, 2004, "Network Management. Concepts and Practice: A Hands-on Approach", Pearson Education, Inc.

[5] Minseok Oh. Network management agent allocation scheme in mesh networks Communications Letters, IEEE Vol.7, Issue12, Dec., 2003, pp.601-603

[6] 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, pp.43-46.

[7] 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, pp.177-181.

[8] Robert Steinke, Micah Clark, Elihu Mcmahon, "A new pattern for flexible worker threads with in-place consumption message queues", Vol.39, Issue2, (April 2005) table of contents pp.71-73 Year of Publication: 2005.

[9] Junaid Ahsenali Chaudhry, and Seung-Kyu Park, Some Enabling Technologies for Ubiquitous Systems, Journal of computer Science 2 (8): 627-633, 2006.

[10] S. Garfinkel, "PGP: Pretty Good Privacy," O'Reily & Associates Inc., 1995.

[11] 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 in 2008-09]

[12] Ma J., Zhao Q., Chaudhary V., Cheng J., Yang L. T., Huang H., and Jin Q.,Ubisafe Computing: Vision and Challenges (I),Springer LNCS Vol.4158, Proc. of ATC-06, 2006.

[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.

[17] 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.

[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 Vol.1, 19-23 April 2004 pp.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] Ilgun, K.; Kemmerer, R.A.; Porras, P.A., "State transition analysis: a rule-based intrusion detection approach," Software Engineering, IEEE Transactions on , Vol.21, No.3, pp.181-199, Mar., 1995.

[22] T. F. Lunt, "Real-time intrusion detection," in Proc. COMPCON, San Francisco, CA, Feb., 1989.

[23] T. F. Lunt et al., "A real-time intrusion detection expert system," SRI CSL Tech. Rep. SRI-CSL-90-05, June, 1990.

[24] T. F. Lunt et al., "A real-time intrusion detection expert system (IDES)," Final Tech. Rep., Comput. Sci. Laboratory, SRI Int., Menlo Park, CA, Feb., 1992.

[25] Radosavac, S.; Seamon, K.; Baras, J.S., "Short Paper: bufSTAT - a tool for early detection and classification of buffer overflow attacks," Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on , Vol., No., pp.231-233, 05-09 Sept., 2005.

[26] 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.

[27] P. J. Denning: Thrashing: Its Causes and Prevention. Proc. AFlPS FJCC 33, 1968, pp.915-922

[28] Turing, Alan M., On Computable Numbers, with an Application to the Entscheidungs Problem. Proceedings of the London Mathematical Society, 2 (42):230-265, 1936.

[29] Hideyuki Takahashi, Takuo Suganuma, Norio Shiratori: AMUSE: An Agent-based Middleware for Context-aware Ubiquitous Services. ICPADS (1) 2005: 743-749.

Junaid Ahsenali Chaudhry, and Seungkyu Park, "AHSEN -- Autonomic Healing-based Self management Engine for Network management in hybrid networks", The Second International Conference on Grid and Pervasive Computing (GPC07), 2007.

Junaid Ahsenali Chaudhry
He received a Ph.D. degree in Computer Science from Ajou Univ. in 2009. He has been a post doctoral research fellow at University of Trento since 2008. His research interests are in the area of Autonomic Computing, Self Healing, Self Growing and Learning Systems, Artificial Intelligence, Distributed Processing and Ubiquitous Networks.