International Journal on Advances in Intelligent Systems

2008 vol. 1 nr. 1

The *International Journal On Advances in Intelligent Systems* is Published by IARIA.
ISSN: 1942-2679
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal On Advances in Intelligent Systems
Volume 1, Number 1, 2008

Editorial Board

First Issue Coordinators

Jaime Lloret, Universidad Politécnica de Valencia, Spain
Pascal Lorenz, Université de Haute Alsace, France
Petre Dini, Cisco Systems, Inc., USA / Concordia University, Canada

Autonomus and Autonomic Systems

- Michael Bauer, The University of Western Ontario, Canada
- Radu Calinescu, Oxford University, UK
- Larbi Esmahi, Athabasca University, Canada
- Florin Gheorghe Filip, Romanian Academy, Romania
- Adam M. Gadomski, ENEA, Italy
- Alex Galis, University College London, UK
- Michael Grottke, University of Erlangen-Nuremberg, Germany
- Nhien-An Le-Khac, University College Dublin, Ireland
- Fidel Liberal Malaina, University of the Basque Country, Spain
- Jeff Riley, Hewlett-Packard Australia, Australia
- Rainer Unland, University of Duisburg-Essen, Germany

Advanced Computer Human Interactions

- Freimut Bodendorf, University of Erlangen-Nuernberg Germany
- Daniel L. Farkas, Cedars-Sinai Medical Center - Los Angeles, USA
- Janusz Kacprzyk, Polish Academy of Sciences, Poland
- Lorenzo Masia, Italian Institute of Technology (IIT) - Genova, Italy
- Antony Satyadas, IBM, USA

Advanced Information Processing Technologies

- Mirela Danubianu, "Stefan cel Mare" University of Suceava, Romania
- Kemal A. Delic, HP Co., USA
- Sorin Georgescu, Ericsson Research, Canada
- Josef Noll, UiO/UNIK, Sweden
- Liviu Panait, Google Inc., USA
- Kenji Saito, Keio University, Japan
- Thomas C. Schmidt, University of Applied Sciences – Hamburg, Germany
- Karolj Skala, Rudjer Bokovic Institute - Zagreb, Croatia
- Chieh-yih Wan, Intel Corporation, USA

- Hoo Chong Wei, Motorola Inc, Malaysia

Ubiquitous Systems and Technologies

- Matthias Bohmer, Munster University of Applied Sciences, Germany
- Dominic Greenwood, Whitestein Technologies AG, Switzerland
- Arthur Herzog, Technische Universitat Darmstadt, Germany
- Reinhard Klemm, Avaya Labs Research-Basking Ridge, USA
- Said Tazi, LAAS-CNRS, Universite Toulouse 1, France

Advanced Computing

- Dumitru Dan Burdescu, University of Craiova, Romania
- Simon G. Fabri, University of Malta – Msida, Malta
- Matthieu Geist, Supelec / ArcelorMittal, France
- Jameleddine Hassine, Cisco Systems, Inc., Canada
- Sascha Opletal, Universitat Stuttgart, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Meikel Poess, Oracle, USA
- Said Tazi, LAAS-CNRS, Universite de Toulouse / Universite Toulouse1, France
- Antonios Tsourdos, Cranfield University/Defence Academy of the United Kingdom, UK

Centric Systems and Technologies

- Razvan Andonie, Central Washington University - Ellensburg, USA / Transylvania University of Brasov, Romania
- Kong Cheng, Telcordia Research, USA
- Vitaly Klyuev, University of Aizu, Japan
- Josef Noll, ConnectedLife@UNIK / UiO- Kjeller, Norway
- Willy Picard, The Poznan University of Economics, Poland
- Roman Y. Shtykh, Waseda University, Japan
- Weilian Su, Naval Postgraduate School - Monterey, USA

GeoInformation and Web Services

- Christophe Claramunt, Naval Academy Research Institute, France
- Wu Chou, Avaya Labs Fellow, AVAYA, USA
- Suzana Dragicevic, Simon Fraser University, Canada
- Dumitru Roman, Semantic Technology Institute Innsbruck, Austria
- Emmanuel Stefanakis, Harokopio University, Greece

Semantic Processing

- Marsal Gavalda, Nexidia Inc.-Atlanta, USA & CUIMPB-Barcelona, Spain
- Christian F. Hempelmann, RiverGlass Inc. - Champaign & Purdue University - West Lafayette, USA
- Josef Noll, ConnectedLife@UNIK / UiO- Kjeller, Norway

- Massimo Paolucci, DOCOMO Communications Laboratories Europe GmbH – Munich, Germany
- Tassilo Pellegrini, Semantic Web Company, Austria
- Antonio Maria Rinaldi, Universita di Napoli Federico II - Napoli Italy
- Dumitru Roman, University of Innsbruck, Austria
- Umberto Straccia, ISTI – CNR, Italy
- Rene Witte, Concordia University, Canada
- Peter Yeh, Accenture Technology Labs, USA
- Filip Zavoral, Charles University in Prague, Czech Republic

International Journal On Advances in Intelligent Systems
Volume 1, Number 1, 2008

Foreword

Finally, we did it! It was a long exercise to have this inaugural number of the journal featuring extended versions of selected papers from the IARIA conferences.

With this 2008, Vol. 1 No.1, we open a long series of hopefully interesting and useful articles on advanced topics covering both industrial tendencies and academic trends. The publication is by-invitation-only and implies a second round of reviews, following the first round of reviews during the paper selection for the conferences.

Starting with 2009, quarterly issues are scheduled, so the outstanding papers presented in IARIA conferences can be enhanced and presented to a large scientific community. Their content is freely distributed from the www.iariajournals.org and will be indefinitely hosted and accessible to everybody from anywhere, with no password, membership, or other restrictive access.

We are grateful to the members of the Editorial Board that will take full responsibility starting with the 2009, Vol 2, No1. We thank all volunteers that contributed to review and validate the contributions for the very first issue, while the Board was getting born. Starting with 2009 issues, the Editor-in Chief will take this editorial role and handle through the Editorial Board the process of publishing the best selected papers.

Some issues may cover specific areas across many IARIA conferences or dedicated to a particular conference. The target is to offer a chance that an extended version of outstanding papers to be published in the journal. Additional efforts are assumed from the authors, as invitation doesn't necessarily imply immediate acceptance.

This particular issue covers papers invited from those presented in 2007 and early 2008 conferences. The papers reflect the evolution of the society from advanced use of the technology for education to user-centric aspects in socio-semantic networks, and complexity of the new environments dealing with adaptive monitoring, load-balancing, and policy-driven autonomic computing.

We hope in a successful launching and expect your contributions via our events.

First Issue Coordinators,
Jaime Lloret, Universidad Politécnica de Valencia, Spain
Pascal Lorenz, Université de Haute Alsace, France
Petre Dini, Cisco Systems, Inc., USA / Concordia University, Canada

International Journal On Advances in Intelligent Systems
Volume 1, Number 1, 2008

CONTENTS

Polling Schedule Optimization for Adaptive Monitoring to Scalable Enterprise Systems

Fumio Machida, Masahiro Kawato, Yoshiharu Maeno
NEC Service Platforms Research Laboratories
1753, Shimanumabe, Nkahara-ku, Kawasaki, Knagawa 211-8666, Japan
{h-machida@ab, m-kawato@ap, y-maeno@aj}.jp.nec.com

Abstract

Adaptive monitoring is a promising technique to automate configurations of a monitoring server in enterprise systems according to the dynamic system reconfigurations such as server scale-out and virtual machine migration. Even after the system reconfiguration, the monitoring server need to be configured properly for providing the fresh information to clients with stabilized server load. In this paper, we propose an adaptive monitoring system that automatically changes the monitoring schedule to satisfy the required freshness under the limited server load after system reconfigurations. The adaptive monitoring system consists of a polling-based monitoring architecture and an algorithm for polling schedule generation. Since the problem for polling schedule generation is classified in NP-hard, we propose an approximation algorithm. According to the results from the experiments with real system reconfiguration scenarios, the adaptive monitoring system improves the variation coefficients of changes of CPU usages and network traffics in the monitoring server by at most 80%. We extend the proposed adaptive monitoring system to be scalable by introducing a hierarchical architecture.

Keywords: Adaptive monitoring, Polling schedule, Virtualization, System reconfigurations, Information freshness

1. Introduction

The emergence of virtual machine technologies enlarges the flexibility of the current enterprise systems. Virtual machine software such as Xen [8], VMware Infrastructure [22] and Microsoft Virtual Center [23] offer a function to create multiple execution environments on a single computer. Enterprise systems can be scale out easily by using virtual machine software and creating a virtual machine on the existing physical environments. System reconfigurations like change of server allocation, server scale out, components replacement and software updates are usually required in common enterprise system administration. Virtual machine can reduce the troubles related to hardware during system reconfigurations because virtual machine does not depend on the physical devices directly.

Although virtual machine enables easy system reconfigurations, frequent system reconfigurations increase administrative operations for the management systems to adapt to the reconfigured target systems. For example, when an administrator adds some virtual machines to the existing systems, he or she has to register the additional targets to monitoring systems or some management tools, and apply appropriate settings. The process of the reconfiguration can be executed automatically by using virtual machines. However, registrations and configuration changes of existing systems need manual operations of administrators. Configuration changes after system reconfigurations are especially important for monitoring systems. Missing registrations and improper setting of monitoring intervals lead to the degradation of the availability and performance of the systems.

We proposed an adaptive monitoring system to reduce administrative operations for reconfigurable enterprise systems. The reduction of the operations for the monitoring settings after system reconfigurations enables easy and speedy adaptation to the target systems. The proposed method generates a monitoring schedule that is a set of monitoring setting satisfying the required freshness of the monitored information and the limited monitoring server load. The system administrator does not need to estimate the impact on the performance and the availability result from the

change of monitoring settings. The schedule generation problem is an integer programming that is classified as NP-hard [7]. If the target system consists of dozens of servers, an optimal schedule is not computable in realistic time. Therefore, we proposed an approximation algorithm for schedule generation. The proposed algorithm generates an optimal schedule under a specific condition. Furthermore, we extend the proposed adaptive monitoring system to be scalable by introducing a hierarchical architecture. A single monitoring server is not realistic for managing thousands of monitoring targets in terms of the load of monitoring server. In the monitoring system using multiple monitoring servers, the query turnaround time and information freshness depend on the number of transit monitoring servers and schedules. To satisfy the requirements from clients for query response time and information freshness, the schedules for multiple monitoring servers need to be optimized. We formulized the problem to decide schedules for multiple monitoring servers configured hierarchically and an approximation algorithm to solve the problem.

The rest of this paper is organized as follows. Section 2 describes the requirements for an adaptive mechanism for monitoring server in enterprise systems. Section 3 presents our adaptive monitoring architecture and an algorithm for polling optimization. Section 4 shows experimental results. Section 5 describes the extension of the adaptive monitoring system and the schedule generation algorithm. Section 6 describes related work and, finally, Section 7 provides the conclusion.

2. Enterprise System Monitoring

Most of enterprise systems have monitoring systems to manage system resources such as servers, network devices, storages and applications. Some commercial products such as HP OpenView Network Node Manager (NNM) [17] and IBM Tivoli NetView [18] provides functions for monitoring resources based on (Simple Network Management Protocol) SNMP [10]. ZABBIX[19], OpenNMS [20] and Nagios [21] come to be known as powerful free monitoring tools that can be used for enterprise-level systems.

Adaptive monitoring appeared in our previous work is a promising technique for enterprise systems to adapt to the change of system configurations and states [1]. The number of monitoring targets in enterprise systems increases and changes dynamically according to the system reconfiguration caused by business requirements and system upgrades. The adaptive monitoring system reduces the administrative operations for monitoring server by automatically optimizes the monitoring configurations at the system reconfigurations. Since virtual machines allow easy system reconfiguration, the concept of adaptive monitoring is especially important in the consolidated server environment using virtual machines.

As a related technique to support the adaptive monitoring, *discovery* is a well-known useful technique to find a newly attached device in the network [9]. NNM provides the discovery function by collecting Address Resolution Protocol (ARP) tables in the target network. If a new server is connected to the target network, the monitoring tool supporting discovery can detect this new target. Although the detection of the new target is automated by discovery, the appropriate configurations for monitoring are up to the administrators. The administrators have to categorize the detected target and set the appropriate monitoring schedule not to have an adverse impact on the existing system.

Our adaptive monitoring system focuses on the quality of the monitoring service, specifically, information freshness and load of monitoring server. Appropriate configurations for monitoring server are important to maintain the quality of monitoring. Freshness is one of the important metrics for quality of resource monitoring [2]. If a monitoring interval is set to a large value, the data stored in the monitoring server is not up to date. The elapsed time from data generation exceeds the required time to live (TTL) and it causes the freshness degradation. To keep the freshness in the required level is important for monitoring aware applications and middleware. The stale (i.e. not fresh) information may cause the incorrect decision and control of monitoring aware applications. The load of the monitoring servers is another quality concern of monitoring systems. Monitoring processes consume system resources such as CPU time and network bandwidth. Excessive processes for information collection in a short time adversely affects system components sharing system resources as well as monitoring server. The processes for information collection need to be scheduled not to gather in a short time period.

3. Adaptive Monitoring System

In this section, we describe an architecture of an adaptive monitoring system and an algorithm for polling schedule generation.

3.1. Architecture

We designed an adaptive monitoring architecture based on the Web Service Polling Engine (WSPE) [2] that is a resource information service for server clusters. WSPE collects resource information from target server nodes via web service protocols, store the information into the temporal cache, and provide the information to the cluster users through the query interface. To keep the fresh information in the cache, WSPE updates the cache repeatedly as per the predefined update schedule. We improved this architecture to reconfigure the update schedule dynamically adapting to the system reconfigurations.

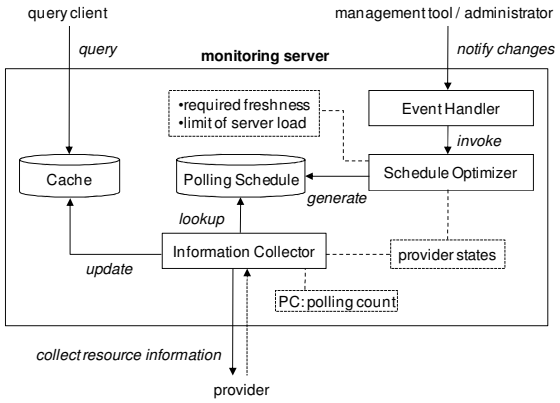


Figure 1. Adaptive monitoring system architecture

Figure1 shows an overview of the proposed monitoring architecture. The monitoring server consists of *Information Collector*, *Schedule Optimizer*, *Event Handler*, *Cache* and *Polling Schedule*. The *Information Collector* collects resource information from providers running on the target servers and updates the *Cache* with collected resource information. All queries from clients are performed on the *Cache*. The availability of each provider is also checked in the information collection process and is managed as provider states. An unavailable resource is dropped from the polling targets. The *Information Collector* counts the *Polling Count* (PC) that indicates the number of occurrences of polling cycles from the start-up. The target information that needs to be updated in one polling cycle is specified in the *Polling Schedule*. The *Polling Schedule* is determined so as to keep the freshness of resource information in the cache and the limit of server load. Since an optimum *Polling Schedule* is changed by the configuration and availabilities of target systems, the *Schedule Optimizer* calculates an optimum *Polling Schedule* in adapting to the latest system configurations. The trigger of schedule optimization is handled by *Event Handler* that receives several notifications about system reconfigurations from management middleware or administrators and determines the needs for schedule optimization. When the *Schedule Optimizer* receives a request for schedule optimization, it identifies the latest system configurations and generates a new *Polling Schedule* by a schedule optimization algorithm that is described in the later section.

The *Polling Schedule* is specified by the Next PC and the Interval PC for each resource as shown in Figure 2. The Next PC specifies the next PC at which to update resource information. When the PC in the *Information Collector* reaches a value of a Next PC, the Information Collector adds this target to the polling targets and collects the latest resource information from the provider. In order to reduce the risk of unexpected peak load caused by polling processes, dispersed values should be used for the Next PCs of different resources. If a large number of target resources have the same value of Next PC, the next polling process has to collect a large amount of resource information in one polling cycle and it may induce a heavy workload on the monitoring server. On the other hand, the Interval PC specifies the number of polling cycles between two consecutive updates. After a polling process to update resource information finishes, the value of the Next PC is calculated by adding the previous value of the Next PC to the Interval PC. The smaller value of Interval PC is preferable to keep the required freshness. The optimum Interval PCs are determined in consideration of the tradeoff between the required freshness and monitoring server load.

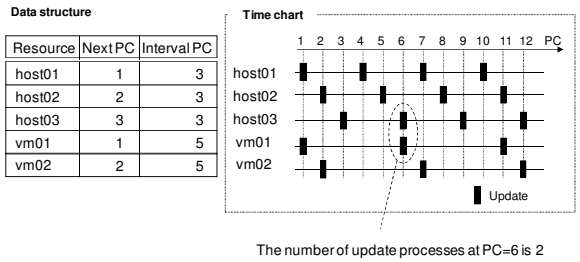


Figure 2. An example of update schedule

The max number of update processes in one cycle of polling must be limited to a certain range of values in consideration of the peak load of the monitoring server. Unexpected peak load called *flush peak* sometimes causes serious system trouble. Since the load of monitoring server depends on the number of target resources having the same Next PC, the peak load of the monitoring server is predictable by the *Polling Schedule* in the proposed system. By optimizing the *Polling Schedule* to keep the number of updates in one

polling cycle in a certain level, we can avoid the risk of the flush peak.

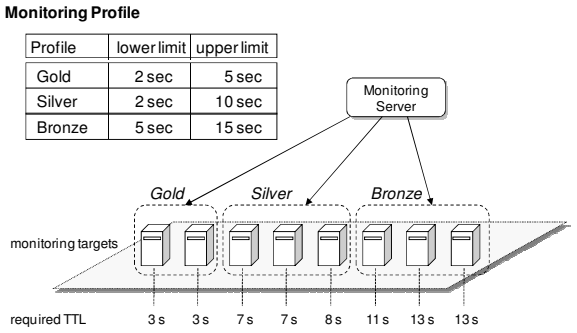


Figure 3. Monitoring profile to group resources

The *monitoring profile* figured in Figure 3 is introduced for grouping the target resources that have the same class of quality level. As the quality of the resource information, the freshness is specified by the TTL in detail. TTL indicates the elapsed time from data generation. The monitoring profile defines the lower limit and the upper limit of the update interval. Since a monitoring profile corresponds to a specific quality level, system administrator create a new monitoring profile when a new quality level is required. Each resource is assigned a monitoring profile and does not belong to the multiple monitoring profiles. Administrators simply manage the allocation of each resource to the specific monitoring profile instead of editing TTL for each resource. By using monitoring profile, the operation for the target addition and the change of monitoring frequency becomes much easier.

3.2. Schedule Generation Problem

The method to generate an optimal polling schedule is an essential part of the adaptive monitoring system. The polling schedule has to satisfy the required freshness of resource information and minimize the number of concurrent updates.

First, the Interval PC for each resource r_i is decided by the allocated monitoring profile p and the current polling interval t_{poll}. The minimum integer j that satisfies the limits defined in the profile is chosen as Interval PC. The Interval PC is expressed as the following expression:

$$\text{IntervalPC}(r_i) = \min\{\, j \mid j \in \mathbf{N}, \text{LL}_p \le t_{\text{poll}} \cdot j \le \text{UL}_p \,\} \tag{1}$$

where LL_p is the lower limit of the update interval for monitoring profile p and UL_p is the upper limit of that.

If any possible values are not found, the administrator should modify the monitoring profile or the polling interval to get a possible Interval PC. Meanwhile, the limited number of concurrent updates (LCU) in a polling cycle is decided in consideration to the acceptable load of the monitoring server.

Next, the Next PC for each resource is decided so that the number of the concurrent updates is not over the LCU. The number of the concurrent updates is changed by each PC and the way to set the Next PC. Since the update processes are executed repeatedly according to each Interval PC, the change in the number of the concurrent updates appears with a period of the least common multiple of Interval PCs (LCMI). We define the polling schedule generation problem as follows.

Problem: Polling Schedule Generation

For each resource information r_i, the update interval PC is defined as $\text{IntervalPC}(r_i) \in \mathbf{N}$. Solve the $\text{NextPC}(r_i) \in \mathbf{N}$ for all r_i, so that the number of concurrent updates is under the LCU at any k from 1 to LCMI.

Solve: $\forall i, \text{NextPC}(r_i)$

Where:

$$\forall k\ (1 \le k \le \text{LCMI}), \quad \sum_{i=1}^{n} \text{U}(k, r_i) \le \text{LCU} \tag{2}$$

$$\text{U}(k, r_i) = \begin{cases} 1 & k - \text{NextPC}(r_i) \equiv 0 \pmod{\text{IntervalPC}(r_i)} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$1 \le \text{NextPC}(r_i) \le \text{IntervalPC}(r_i) \tag{4}$$

The schedule generation problem is an integer programming of $\text{NextPC}(r_i)$, that is classified as NP-hard. It takes exponential time of the number of targets "n" to decide if any possible schedule exists or not. If there are a large number of targets in the system, the above problem cannot be solved in practical time.

3.3. Schedule Generation Algorithm

To solve the schedule generation problem in practical time, we propose an algorithm by using an approximate method.

Algorithm 1:

1) Make groups that have the same value of $\text{IntervalPC}(r_i)$.

$$G_j = \{r_i \mid \text{IntervalPC}(r_i) = j\} \tag{5}$$

Define J as a set of possible values as j.

2) For each group, generate schedule that minimizes the concurrent updates. Label all r_i in G_j as $r_{i,k}$ $(1 \le k \le |G_j|)$ and set the NextPC($r_{i,k}$) based on this label.

$$\text{NextPC}(r_{i,k}) = k \ (\text{mod IntervalPC}(r_i)) \quad (6)$$

The number of max concurrent updates for G_j is calculated by: $\left\lceil \frac{|G_j|}{j} \right\rceil$

3) Combine all generated schedules and calculate sum of the number of concurrent updates.

$$\sum_{j \in J} \left\lceil \frac{|G_j|}{j} \right\rceil \quad (7)$$

Compare the sum of the number of concurrent updates to the LCU. If the sum of the number of concurrent updates is smaller than LCU, output the generated schedule as a possible schedule. Otherwise, give up the schedule generation.

Algorithm 1 divides the all r_i into the groups that have the same value of IntervalPC(r_i) and solves the partial optimal schedule for each group. By gathering the partial schedules, the max number of concurrent updates is minimized in most situations. Furthermore, the algorithm always outputs a result in $O(n)$ time.

If each pair of IntervalPC(r_i)s of the different groups is relatively prime, the Algorithm 1 always solves the optimal schedule (i.e. minimize the number of the concurrent updates) by the following theorems.

Theorem 1:

When all of the IntervalPC(r_i) have the same value, the max number of the concurrent updates of the schedule is equal to or more than $\left\lceil \frac{n}{\text{IntervalPC}(r_i)} \right\rceil$, where n is the number of targets.

Proof 1:

Let α be the max number of the concurrent updates. All of r_i have to be updated during IntervalPC(r_i) within α update processes.

$$n \le \alpha \cdot \text{IntervalPC}(r_i) \quad (8)$$

Because α is an integer value, the following condition is obtained.

$$\alpha \ge \left\lceil \frac{n}{\text{IntervalPC}(r_i)} \right\rceil \quad (9)$$

■

Theorem 2:

G_p and G_q are groups of resource information that has intervals of p and q. If p is coprime to q, the max number of the concurrent updates of the update schedule for all elements of G_p and G_q is equal to or more than $\left\lceil \frac{|G_p|}{p} \right\rceil + \left\lceil \frac{|G_q|}{q} \right\rceil$.

Proof 2:

For any $r_{p1} \in G_p$ and any $r_{q1} \in G_q$, the PC to update: $t_p(r_{p1})$ and $t_q(r_{q1})$ are generally represented by:

$$t_p(r_{p1}) = m_p \cdot p + \text{NextPC}(r_{p1}) \quad (10)$$

$$t_q(r_{q1}) = m_q \cdot q + \text{NextPC}(r_{q1}) \quad (11)$$

where, m_p and m_q are any positive integer values.

Here, for any NextPC(r_{p1}) and any NextPC(r_{q1}), there exists a pair of m_p and m_q satisfying $t_p(r_{p1}) = t_q(r_{q1})$ modulo pq. This is derived from the *Chinese remainder theorem* [6].

Therefore, there exists a case where the number of concurrent updates is 2 for any pair of r_{p1} and r_{q1}. The max number of the concurrent updates, α, is given by:

$$\alpha = \alpha_p + \alpha_q \quad (12)$$

where α_p and α_q are the max number of the concurrent updates for G_p and G_q.

From the Theorem 1, the following condition is obtained.

$$\alpha \ge \left\lceil \frac{|G_p|}{p} \right\rceil + \left\lceil \frac{|G_q|}{q} \right\rceil \quad (13)$$

■

Because the max number of the concurrent updates of the schedule generated by the Algorithm 1 is $\sum_{j \in J} \left\lceil \frac{|G_j|}{j} \right\rceil$, the output schedule is always optimal if each pair of IntervalPC(r_i)s of the different groups is relatively prime.

4. Evaluation

This section describes the experimental evaluations of the proposed adaptive monitoring system using a system reconfiguration scenario.

4.1. Monitoring load estimation

The load of the monitoring server such as CPU usage and the amount of the network traffic depends on the number of concurrent update processes. By investigating the relationship between the load of the monitoring server and the number of the concurrent updates, the load of the monitoring server at real

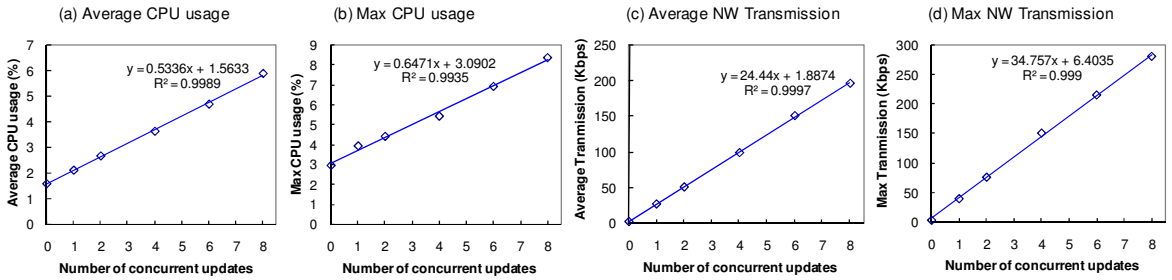


Figure 4. The relationship between the number of concurrent updates and the monitoring loads

execution can be estimated from the installed polling schedule.

The experimental environment has a monitoring server that has 3GHz Intel Pentium4 processor and 2.3 GB of RAM. On this server, WSPE collects resource information from several physical and virtual machines. Each target provides 12 KB of resource information. All nodes used in the experiments are connected by 100 Mbps ethernet.

In this testing environment, we measured several system metrics like CPU usages, memory usages, disk I/O and network traffics by varying the number of concurrent updates. The relationship between the system metrics and the number of concurrent updates can be characterized by regression analysis. Figure 4 (a) shows the plots of the measured values of CPU usages under the limited number of the concurrent updates. The relationship is expressed as the following expression by applying the least square method to the observed values.

$$y = 0.5336 \cdot x + 1.5633 \qquad (14)$$

where x is the number of concurrent updates in a polling cycle and y is the average CPU usage. The regression coefficients change depending on the resource capacities and states of usage. For example, the more CPU power the monitoring server can use, the smaller value the gradient of the regression line for the CPU usage. As far as this experimental environment is used, the average CPU usage of the monitoring server is predictable by the obtained regression formula. In addition to the average CPU usage, the max CPU usage, the average and max network transmission traffic also have the linear relation with the number of concurrent updates (see Figure 4). The other performance data such as network receive traffic, memory usage and disk I/O does not have linear relationship with the number of concurrent updates in our testing environment. From the results of this investigation, we can find an appropriate value of the number of concurrent updates to keep the load of monitoring server in a certain level.

4.2. Adaptation to system reconfigurations

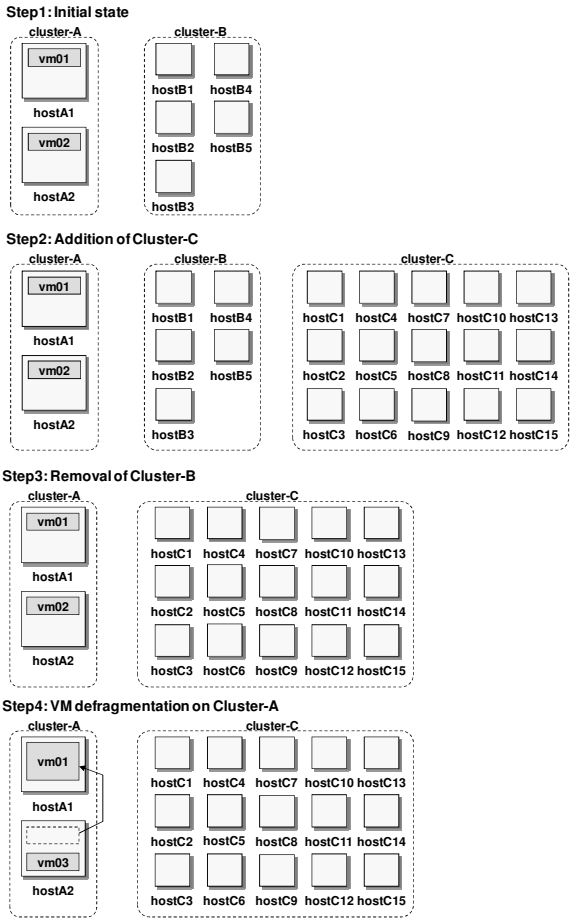


Figure 5. VM defragmentation scenario

The monitoring adaptation mechanism was evaluated by a scenario involving the virtual machine defragmentation as depicted in Figure 5. The monitoring setting is automatically changed by the proposed adaptation mechanism for each step of the

scenario. The experimental environment consists of three different clusters, cluster-A, cluster-B and cluster-C. The cluster-A is established on the virtualized environment using Xen 2.0 on Fedora Core 4. Cluster-B consists of 5 nodes and Cluster-C has 15 nodes.

In the first step of the scenario (step 1), the cluster-A and the cluster-B are monitored from the monitoring server running on a management server. In the second step (step 2), the cluster-C is added to the monitored target of the monitoring server. In the third step (step 3), the cluster-B is removed from the monitored target. In the final step (step 4), the defragmentation of virtual machines on the cluster-A is performed. The defragmentation moves the virtual machine instance vm02 to the hostA1, then merges instances of vm01 and vm02, and finally starts a new virtual machine instance vm03 in the created resource space on the hostA2. In this experiment, the merge process simply stops the vm02 and expands the resource allocation to vm01.

All physical servers and virtual machines have corresponding monitoring profiles. Table 1 shows the four different monitoring profiles used in the experiments. The polling interval t_{poll} is set to 1 second and the value of LCU is set to 8. For each step of the scenario, the optimization algorithm generates the optimal update schedule that meets the conditions specified in monitoring profiles and minimizes the number of concurrent updates under LCU. The generated update schedules for each step are shown in Table 2.

Besides the *optimization* approach, the *simple polling* approach and the *without-optimization* approach were also evaluated by this scenario for the sake of comparison. The simple polling approach updates all of information at regular intervals like SNMP polling. The regular interval was set to 10 seconds. The without-optimization approach updates resource information at specific intervals requested from each monitoring profile. Although this approach satisfies the conditions of the monitoring profiles, the number of concurrent updates is not bounded.

Table 1. Monitoring profiles

	lower limit	upper limit
Platinum	3s	10s
Gold	5s	15s
Silver	7s	20s
Bronze	11s	30s

Table 2. Update schedules for each step

cluster	node	profile	Interval PC	Next PC			
				Step 1	Step 2	Step 3	Step 4
A	hostA1	Platinum	3	1	1	1	1
	hostA2	Platinum	3	2	2	2	2
	vm01	Bronze	11	1	1	1	1
	vm02	Bronze	11	2	2	2	
	vm03	Bronze	11				2
B	hostB1	Platinum	3	3	3		
	hostB2	Platinum	3	1	1		
	hostB3	Platinum	3	2	2		
	hostB4	Platinum	3	3	3		
	hostB5	Platinum	3	1	1		
C	hostC1	Gold	5		1	1	1
	hostC2	Gold	5		2	2	2
	hostC3	Gold	5		3	3	3
	hostC4	Gold	5		4	4	4
	hostC5	Gold	5		5	5	5
	hostC6	Gold	5		1	1	1
	hostC7	Gold	5		2	2	2
	hostC8	Gold	5		3	3	3
	hostC9	Gold	5		4	4	4
	hostC10	Gold	5		5	5	5
	hostC11	Silver	7		1	1	1
	hostC12	Silver	7		2	2	2
	hostC13	Silver	7		3	3	3
	hostC14	Silver	7		4	4	4
	hostC15	Silver	7		5	5	5

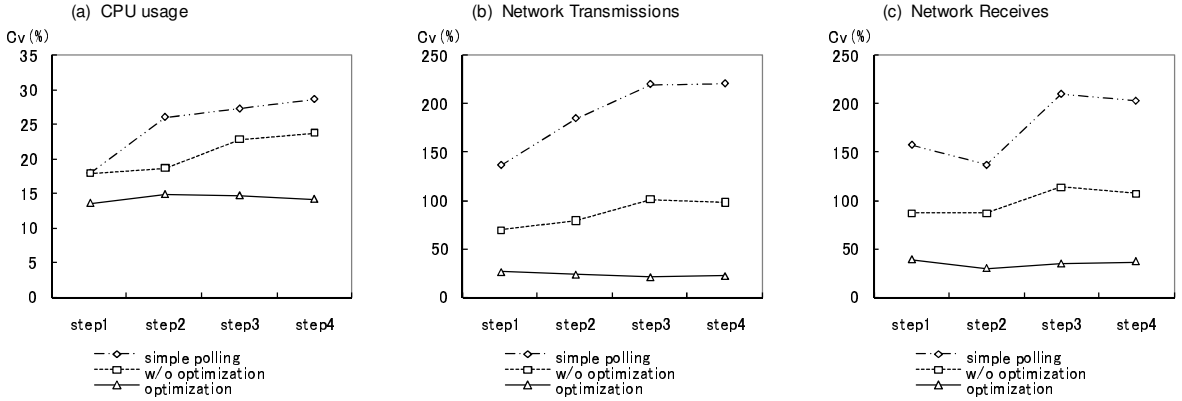


Figure 6. Variation coefficients of CPU usages and network traffics

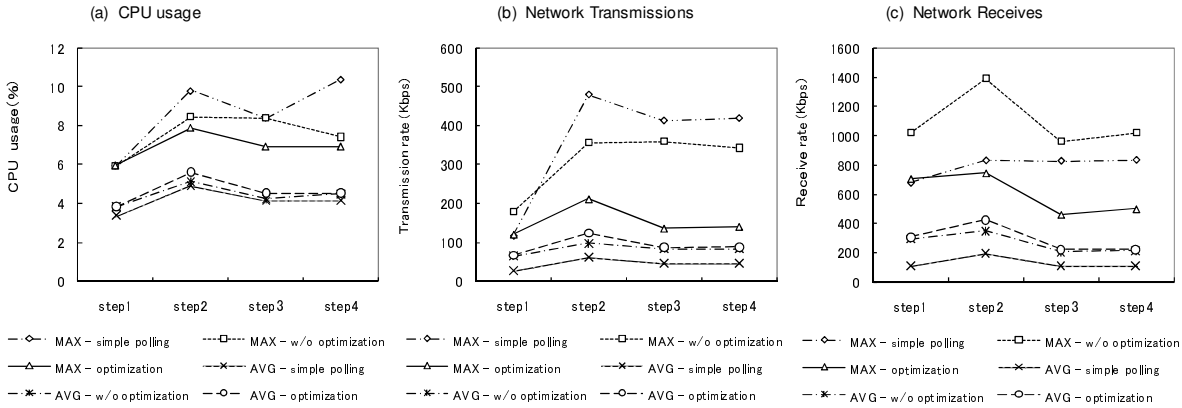


Figure 7. Max and average values of CPU usages and network traffics

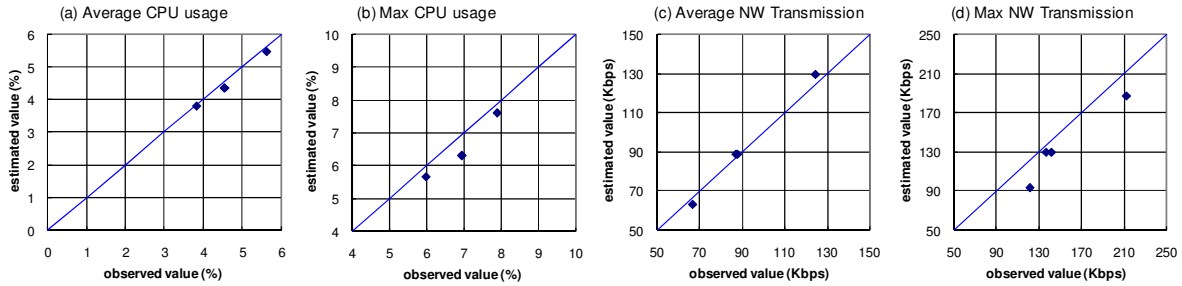


Figure 8. Observed values versus estimated values by regression functions

We observed the variation coefficients of CPU usages and network traffics for each step of the scenario (see Figure 6). All these variation coefficients were calculated from the time-series performance data of three minutes duration in each step. The variation coefficient of optimization approach is the lowest in any case and the values do not change significantly over the steps. Compared to the without-optimization approach, the variation coefficient of network transmission traffic is reduced by 80% at step 3 (see Figure 6 (b)). The results indicate that the proposed adaptive monitoring system stabilize the load of monitoring server by optimizing the polling schedule according to the system reconfigurations.

Meanwhile the max values of CPU usages and network traffics during the three minutes for each step are shown in Figure 7. The results provide a study of risk for flash peak of the resource usage. The optimization approach can lower down the max values of CPU usages and the network traffics by dispersing the update processes over time. Compared to the without-optimization approach, the max transmission traffic is reduced by 62% at step3 (see Figure 7 (b)). The proposed optimization approach reduces the risk of the flash peak.

Additionally the approximate max values are predicted by using the regression function described in Section 4.1. Figure 8 shows the relationship between the measured values and estimated values. The results show that the estimation provides a good indicator for availability of the monitoring server.

5. Scalable adaptive monitoring

In this section, we extend the adaptive monitoring system to hierarchical configurations. To satisfy the requirements for TTLs from lots of clients, we propose an algorithm for multiple schedules generation.

5.1. Requirements for scalable monitoring

Large scale enterprise systems distributed in multiple locations have thousands of monitoring targets such as servers, routers, switches and applications. A single monitoring server is not enough to collect the resource information from thousands of monitoring targets from the concern for the load of monitoring

server and network. Generally, for such a large-scale system, multiple monitoring servers are configured hierarchically to integrate the resource information. HP's NNM can manage 25000 of devices by organizing monitoring servers hierarchically. MDS [15] and Ganglia [16] support hierarchical architecture to aggregate resource information from thousands of nodes in the grid environment.

Although the hierarchically architecture improves the scalability of monitoring systems, the overhead of multiple monitoring servers degrades the query performance and freshness of resource information. Users and applications using the monitored information require the specific level of the query performance and information freshness. Configurations for monitoring servers for satisfying the quality requirements are much more complex than the case with a single server. Adaptive monitoring that reduces the manual operations for monitoring settings after system reconfiguration is also valuable in the large scale enterprise systems.

For the large scale enterprise systems, we extend the WSPE to hierarchical configurations. Each WSPE handles the event of system reconfigurations and adapts the polling schedule automatically to the target systems. By optimizing the polling schedule in each WSPE, all of requirements are satisfied under the capacity limitations of monitoring servers.

5.2. Hierarchical configuration of WSPEs

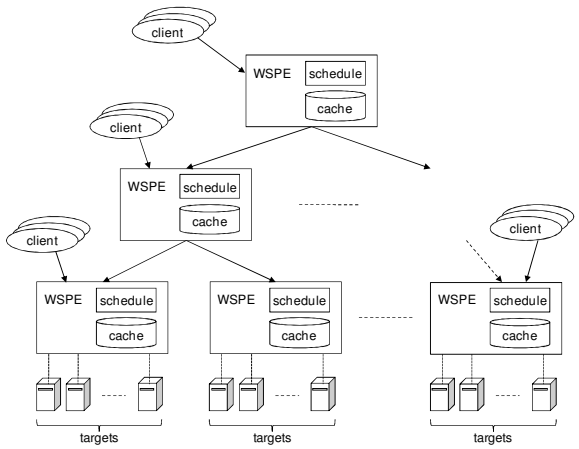


Figure 9. Hierarchically-configured WSPEs

Figure 9 shows a hierarchical configuration of WSPEs to collect resource information from widely-distributed systems. Each WSPE has own polling schedule to keep the freshness of the resource information in the cache. Some WSPEs collect resource information from the other WSPE instead of collecting directly from the target resources. It reduces traffics to the target resources and distributes the load of monitoring servers. Clients query the resource information to the nearest WSPE that has the target information in the cache. The query response time is estimated by the turnaround time from the client to the nearest WSPE.

The TTL of resource information r_i in the query results depends on the polling intervals of all WSPEs on the path from the client c_h to the target resource r_i. Here we denote the polling interval for resource r_i in the WSPE w_j as $t_{\text{poll}}(w_j, r_i)$. Let $W_{h,i}$ be the set of WSPEs on the path from the client c_h to the target resoruce r_i. The TTL of resource information r_i for the client c_h is bounded as the following expression:

$$t_{\text{TTL}}(c_h, r_i) \le t_{\text{resp}}(c_h, w^1, r_i) + \sum_{w_j \in W_{h,i}} t_{\text{poll}}(w_j, r_i) \qquad (15)$$

where $t_{\text{resp}}(c_h, w^1, r_i)$ is the time taken to deliver the information r_i from the nearest WSPE $w^1 \in W_{h,i}$ (see Figure 10).

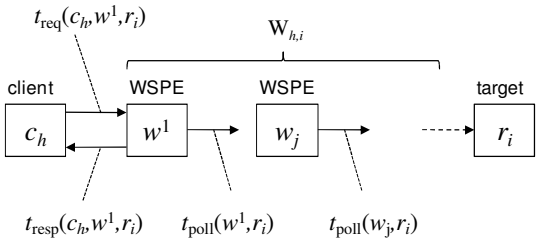


Figure 10. Model of hierarchical WSPEs

Let $t_{\text{req}}(c_h, w^1, r_i)$ be the time taken to request the query for r_i from c_h to w^1. The query response time is expressed as follows:

$$t_{\text{query}}(c_h, w^1, r_i) = t_{\text{req}}(c_h, w^1, r_i) + t_{\text{resp}}(c_h, w^1, r_i). \qquad (16)$$

If the $t_{\text{query}}(c_h, w^1, r_i)$ does not meet the required performance of c_h due to the limitations of network performance or server capacity, an additional placement of a WSPE near the client improves the query performance at the expense of the information freshness.

We assume the number of WSPEs and networks are given by the requirements for the query response time of each client and the limitation derived from the network topology. We discuss the problem of polling schedule optimization to guarantee the required TTLs for all clients under the limitations of server loads.

5.3. Multiple Polling Schedules Generation

Polling schedules for all WSPEs need to be optimized for satisfying the requirements for TTL of

resource r_i from the client c_h: RTTL(c_h, r_i) under the limitation of server loads given by the LCU of each WSPE. For a single WSPE, the Interval PCs are determined by the formula (1) based on the monitoring profiles. However, for the hierarchically-configured WSPEs where many clients request to guarantee the TTL of resource information, the Interval PCs need to be determined by considering the requested RTTLs and polling intervals of other WSPEs.

The problem to solve the polling schedules of WSPEs under the conditions about RTTLs and LCUs is defined as follows.

Problem: Multiple Polling Schedules Generation

Solve the IntervalPC(w_j, r_i) and NextPC(w_j, r_i) for each WSPE w_j to satisfy all requirements of RTTL(c_h, r_i), under the limitations of the number of concurrent updates LCU(w_j).

$$\text{Solve: } \forall i, \forall j, \quad \text{IntervalPC}(w_j, r_i), \text{NextPC}(w_j, r_i)$$

Where:

$$\forall h, \quad t_{\text{TTL}}(c_h, r_i) \le \text{RTTL}(c_h, r_i) \tag{17}$$

$$\forall k, \quad \sum_{i=1}^{n} \text{U}(k, w_j, r_i) \le \text{LCU}(w_j) \tag{18}$$

$$\text{U}(k, w_j, r_i) = \begin{cases} 1 & k - \text{NextPC}(w_j, r_i) \equiv 0 \ (\text{mod IntervalPC}(w_j, r_i)) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$1 \le \text{NextPC}(w_j, r_i) \le \text{IntervalPC}(w_j, r_i) \tag{20}$$

Constraint (17) states the limitation from the requirements for RTTL(c_h, r_i). Constraints (18) and (19) state the limitation of the LCUs. The problem of multiple polling schedules generation is an integer programming and NP-hard as well as the schedule generation problem discussed in Section 3.2.

5.4. Multiple Schedules Generation Algorithm

We propose an algorithm to generate multiple polling schedules satisfying the requirements of RTTLs and the limitations of LCUs for hierarchically-configured WSPEs. The proposed algorithm generates polling schedules satisfying the constraints (18) by applying algorithm 1 for each WSPE and readjusts the Interval PCs so as to satisfy the constraints (17) by changing the assignment of monitoring profiles.

Algorithm 2:

1) Generate polling schedules for all w_j by applying algorithm 1 with the default monitoring profiles and the limitation of LCU that are set in each WSPE.
2) For all requirements for TTL of resource r_i from the client c_h, check if the max value of $t_{\text{TTL}}(c_h, r_i)$ calculated by (15) is below the RTTL(c_h, r_i). If all RTTLs are satisfied, output the schedules and finish the schedule generation process. Otherwise, go to the following steps to readjust the polling schedules.
3) Let w^k $(1 \le k \le |\text{W}_{h,i}|)$ be the sequence of WSPEs on the path to the r_i from c_h. The sequence starts from w^1 that is the nearest WSPE from c_h. In the sequence, search a w^k that can readjust schedule so as to satisfy the requirements of RTTL(c_h, r_i) by the following step 4. If the w^k that can readjust schedule is not found by the iteration of step 4, give up the multiple schedule generation.
4) In the given w^k, for resource r_i, change the allocation of profile that satisfies both of the following conditions.

$$\text{LL}_p \le \text{RTTL}(c_h, r_i) - t_{\text{TTL}}(c_h, r_i) + t_{\text{poll}}(r_i)$$
$$\text{UL}_p \ge \text{RTTL}(c_h, r_i) - t_{\text{TTL}}(c_h, r_i) + t_{\text{poll}}(r_i) \tag{21}$$

where LL_p is the lower limit of the update interval for monitoring profile p and UL_p is the upper limit of that. If any profile p that satisfies the conditions (21), calculate a new IntervalPC(w_j, r_i) by the expression (1) with the new profile and generate a schedule by the algorithm 1. Repeat finding the possible profiles until get the schedule or check all profiles.

Since the algorithm 2 is an approximation algorithm, it does not always output the multiple polling schedules even if there is a possible solution. However, the algorithm can change the polling schedules locally to satisfy the requirements of RTTL(c_h, r_i) instead of globally optimization. The algorithm gives the advantage to adapt the existing polling schedules to the change of RTTL(c_h, r_i). Since the monitoring profiles are edited by system administrator as necessary, the number of monitoring profiles is limited. The routine of step 4 is processed in the finite execution time.

6. Related work

Scalable performance monitoring systems have been well studied in the context of grid computing. A white paper summarized and evaluated lots of presented grid monitoring systems [13]. Some advanced monitoring systems such as Remos [11] and Network Weather Service (NWS) [12] have a function

to forecast the performance changes. In contrast to several existing works for the grid monitoring systems, we focus on the quality of the monitoring service, namely freshness of resource information, in the large scale enterprise systems.

The quality of monitoring is important especially in the grid and autonomic computing. The monitoring requirements differ across applications hosted on the server and change over time corresponding to the system configurations. QMON [4] provides a function to classify and configure the quality of monitoring based on service level agreement (SLA). QMON changes the monitoring configuration dynamically by using the concept of "monitoring channel". However, the current QMON does not support the adaptation mechanism to the target system reconfiguration such as server addition and deletion.

Although freshness is important for applications using monitored data, the significant emphasis on the freshness results in a "flash crowd" caused by monitoring processes [14]. The monitoring system must manage the server load to avoid the flash crowd. Our experimental results show that the flash crowd is avoidable by the optimized schedule.

For the network management, an efficient polling technique for SNMP is proposed [5]. This technique provides a function to minimize the polling queries to the SNMP agents by using the usage parameters defined by the applications. However, any method to avoid the flash crowd is not supported.

The necessity of the polling optimization is also described in the grid monitoring system using slacker coherence model [3]. The slacker coherence model is useful to minimize the polling with consideration to the out-of-sync period of the data. Although this model considers the load of the target nodes, the server-side load is not considered. Therefore, there is no guarantee that the flash crowd does not occur.

7. Conclusion

This paper proposed the adaptive monitoring system to reduce the administrative operations in the large-scale enterprise systems. The monitoring server guarantees the freshness of resource information in the cache by the polling based cache updates. The update processes are scheduled to satisfy the requirements of freshness and the limitation of monitoring server load. We presented a schedule generation algorithm and proved that the algorithm generates an optimal schedule minimizing the max number of concurrent updates. From the experimental results, the variation coefficients of CPU usages and network traffics are improved by at most 80%, and the max values at the load peak are decreased by at most 62%. The results show that the proposed method can stabilize the load of monitoring server and can reduce the risk of flash peak according to the current system configuration. We presented as well the extension of the adaptive monitoring system to be scalable with the algorithm for generating multiple polling schedules. By applying the proposed algorithm to hierarchically-configured WSPEs, we can guarantee all requirements for freshness of resource information from multiple users under the limited loads of monitoring servers.

References

[1] F. Machida, M. Kawato and Y. Maeno, Adaptive Monitoring for Virtual Machine Based Reconfigurable Enterprise Systems, 3rd International Conference on Autonomic and Autonomous Systems (ICAS2007), 2007.

[2] F. Machida, M. Kawato and Y. Maeno, Guarantee of Freshness in Resource Information Cache on WSPE: Web Service Polling Engine, 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2006), 2006.

[3] R. Sundaresan, M. Lauria, T. Kurc, S. Parthasarathy and Joel Saltz, Adaptive Polling of Grid Resource Monitors Using a Slacker Coherence Model, 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03), 2003.

[4] S. Agarwala, Y. Chen, D. Milojicic and K. Schwan, QMON: QoS- and Utility-Aware Monitoring in Enterprise systems, 3rd IEEE International Cofference on Autonomic Computing (ICAC2006), 2006.

[5] M. Cheikhrouhou and J. Labetoulle, An Efficient Polling Layer for SNMP, IEEE/IFIP Network Operations and Management Symposium (NOMS2000), 2000.

[6] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd Edition, Section 4.3.2, page 286, Addison-Wesley, 1997.

[7] B. Korte, J. Vygen, Combinatorial Optimization: Theory and Algorithms, Japanese Edition 2005, Section 15.7 NP-Hard Problems, Springer, 2005.

[8] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham and R. Neugebauer, Xen and the Art of Virtualization, 19th ACM Symposium on Operating Systems Principles (SOSP19), 2003.

[9] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, Topology discovery in heterogeneous IP networks, 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM2000), 2000.

[10] J. Case, M. Fedor, M. Schoffstall and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, 1990.

[11] P. Dinda, T. Gross, R. Karrer, B Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland, The architecture of the remos system. In 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), August, 2001.

[12] R. Wolski, Experiences with Predicting Resource Performance On-line in Computational Grid Settings, ACM SIGMETRICS Performance Evaluation Review, Volume 30, Number 4, March, 2003, pp 41--49.

[13] M. Gerndt, R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorzki, H. Truong, T. Fahringer, M. Bubak, E. Laure and T. Margalef. Performance Tools for the Grid: State of the Art and Future, Automatic Performance Analisis: Real Tools White Paper, 2004.

[14] R. Desai, S. Tilak, B. Gandhi, M. J. Lewis and N. B. Abu-Ghazaleh, Analysis of Query Maching Criteria and Resource Monitoring Models for Grid Application Scheduling, 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2006), 2006.

[15] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, Grid Information Services for Distributed Resource Sharing, In Tenth IEEE International Symposium on HighPerformance Distributed Computing (HPDC10), IEEE Press, August 2001.

[16] M. L. Massie, B. N. Chun, and D. E. Culler, The Ganglia Distributed Monitoring System: Design, Implementation, and Experience, Parallel Computing, Vol. 30, Issue 7, July, 2004.

[17] HP OpenView Network Node Manager (NNM): http://h20229.www2.hp.com/products/nnm/index.html

[18] IBM Tivoli NetView: http://www-306.ibm.com/software/tivoli/products/netview/

[19] ZABBIX: http://www.zabbix.org/

[20] OpenNMS: http://www.opennms.org/index.php/Main_Page

[21] Nagios: http://nagios.org/

[22] VMware: http://www.vmware.com/

[23] Microsoft Virtual Center: http://www.microsoft.com/windowsserversystem/virtualserver/

Effective Design of Trust Ontologies for Improvement in the Structure of Socio-Semantic Trust Networks

Nima Dokoohaki[1], Mihhail Matskin[2]

***Abstract*—Social ecosystems are growing across the web and social trust networks formed within these systems create an extraordinary test-bed to study relation dependant notions such as trust, reputation and belief. In order to capture, model and represent the semantics of trust relationships forming the trust networks, main components of relationships are represented and described using ontologies. This paper investigates how effective design of trust ontologies can improve the structure of trust networks created and implemented within semantic web-driven social institutions and systems. Based on the context of our research, we represent a trust ontology that captures the semantics of the structure of trust networks based on the context of social institutions and ecosystems on semantic web.**

***Index Terms*—Semantic Trust, Trust Networks, Trust Ontology, Semantic Social Networks, Ontology Engineering, Structural Analysis.**

I. INTRODUCTION

Semantic web is described to be a web of knowledge having properties such as heterogeneity, openness and ubiquity. In such an environment where everyone has the ability to contribute, trustworthiness of these people and their contributions are of great importance and value. As stressed, trust plays a crucial role in bringing the semantic web to its full potential.

A trust network can be seen as a structure capturing metadata on a web of individuals with annotations about their trustworthiness. Considering social network as our context, a trust network can be seen as an overlay above the social network that carries trust annotations of the metadata based on the social network, such as user profiles and information. Social networks are gaining increasing popularity on the web while semantic web and its related technologies, are trying to bring social networks to their next level. Social networks are using the semantic web technologies to merge and integrate the social networking user profiles and information. Such efforts are paving the path toward semantic web-driven social ecosystems. Merging and integrating social networking data and information can be of business value and use to web service consumers as well as to web service providers of social systems and networks. Ontologies, at the core of semantic-web driven technologies lead the evolution of social systems on the web. Describing trust relations and their sub-components using ontologies, creates a methodology and mechanism in order to efficiently design and engineer trust networks.

"Structure of a given system is the way by which their components interconnect with no changes in their organization" according to [1]. Determining the structure of a society of agents on a trust network structure within a semantic social system, can help us determine the organizational structure of a system. Having this capability we can determine an organization's certain factors such as flexibility, change capacity, etc.

In this paper we investigate how effective design of trust ontologies can improve the structure of trust networks created and implemented within semantic web-based social systems. To address the efficient design of trust networks on semantic web-driven social systems, we have engineered and analyzed a trust ontology [2]. Our trust ontology is based on the main concept of *Relationship*, that models the main element of trust networks, and two concepts of *Main Properties* and *AuxiliaryProperties*, which model properties of relationships.

In order to effectively design an ontology for trust, we have introduced a framework for comparing and evaluating trust ontologies. As an experiment, several ontologies of trust have been evaluated according to our framework. To understand the process of engineering the ontology itself, all phases and steps taken during the process of building our proposed trust ontology have been mentioned in details. As an experiment, we have studied the structure of the trust network to describe how a trust ontology can serve as the framework for engineering efficient and scalable trust networks. Same experiment data have been used to create network of other similar works structure-wise to get a deeper knowledge of the

[1]Nima Dokoohaki is a PhD candidate at Department of Electronics, Computer and Software Systems (ECS), School of Information and Communications Technology (ICT), Royal Institute of Technology (KTH), Stockholm, Sweden. Email: nimad@kth.se

[2]Mihhail Matskin is a professor at Department of Electronics, Computer and Software Systems (ECS), School of Information and Communications Technology (ICT), Royal Institute of Technology (KTH), Stockholm, Sweden and professor II (adjunct professor) at Norwegian University of Science and Technology, (NTNU), Trondheim, Norway. Email: misha@imit.kth.se

network structure with respect to ontology design disciplines. The contents of this paper are organized as follows: following the background study and discussion on related research in section 2, state of art in trust ontologies is presented in section 3, our trust ontology is introduced in the section 4, in section 5 trust networks analysis is presented and discussed. Finally we conclude in section 6 and we discuss the future research in section 7.

II. BACKGROUND

Within the context of social semantic systems, there has been an extensive amount of efforts based on both academic and practical approaches in order to design and engineer trust networks, but none of the existing works in the field were designed bearing structural and design issues in mind. In this section we introduce the technologies that we have incorporated and considered in our approach.

We divide the foundation of our work into two main topics, namely: semantic social networks and trust. In this section we also give a detailed and thorough overview into each field. Each overview is divided into subsections where each of the substantial topics is further studied and discussed.

A. Socio-Semantic Ecosystems Overview

In 1967, Stanley Milgram introduced "Small World Hypothesis" [3], which was published by American Sociologist. Social networks became popular in 1990s. A social network is generically defined as a set of people gathered together through connections or links, according to [5].

Web has become a ground for bringing the notion of society of people into life. A web-driven social network needs to be accessible using a web browser and within this network people should be able to explicitly (or implicitly) state their connections and their links to individuals or group of individuals, according to [21].

Web-based social networks continue to evolve, while what is most important today is that connections on these networks, are not single dimensional anymore and today you can model and state different aspects of relationships, such as trust.

In 2005, according to [21], there were 115,000,000 accounts within social networks scattered across about 18 online networking communities. It's important to consider that not all these accounts correspond to a single individual. Many people have multiple memberships across multiple networks, at the same time.

Size of the social networks will continue to grow everyday as people realize the "hidden" values of social networking day by day [8]. This growth will continue in size aspect of web grounded social networks and will not stop and as many have predicted [7] [8], the so called "email scenario" will take place, where the number of advertisements and SPAM messages will increase so drastically that by some point of time these networks will literally collapse.

There is a strong and growing demand for fusion of the data from different social networks on web. Many are interested in sharing their profiles, while others are interested in merging their data from multiple networks.

Two main reasons can be stated and discussed here:

First and foremost, great amount of this data which is scattered throughout all these sites are not shareable and are inaccessible from other networks. Second, as stated many users have different accounts across different networks and if their data merge, then many of these accounts might become a single account.

In addition to individuals and users on the web, social networks have become the target of the businesses and industries. There are many businesses and enterprises which sell packages of social networking capable software to their users. So the value of social networking exceeds beyond the borders of individuals and businesses now.

1) Vision of semantic web-driven social institutions

Social metadata fusion, in the form of sharing or integration brings business value to entities living within such ecosystems. The vision of "Semantic Social Network (SSN)" [4], describes the fusion and integration of social data across social networks, located on a web of semantics.

This vision is based upon two important dimensions:

First, semantic descriptions of social data about people available on the web in public, expressed in a formal metadata language such as XML or RDF, with explicitly described links to other people on same or different networks.

Second, semantic references to those descriptions described and stored in a formal metadata language such as RDF or XML [4] [5].

There were several attempts to bring this vision into life. One of the most important and influential ones is FOAF (friend-of-a-friend) project [6].

2) FOAF and SIOC: bringing the vision into life

FOAF project creates an RDF vocabulary for describing people and the relationship between them. In this way it can be used as the "glue" in between semantic web and social ecosystems, according to [10].

As described, current Web communities are distributed all around the web, with no links in between them, according to [11].

In order to bring semantics to online communities, SIOC [12] (Semantically-Interlinked Online Communities) tries to create the so called "glue" through SIOC ontology [13] [14].

SIOC aims to enable the integration of Web community information and creates the possibility of describing and presenting the social web of data using RDF. We can think of FOAF as an enabler for describing semantic web of individuals, while SIOC enables describing semantic web of communities of individuals.

SIOC utilizes the FOAF vocabulary for expressing personal profile and social networking information [11].

3) Modeling social networks on semantic web

Social Network Analysis (SNA) [15] [16], is the science of studying and analyzing a networked setting and it has been applied to settings of networks of health, innovation, etc. Network analysis provides the theoretical as well as practical background for studying how to analyze the network participation effect on certain grounds such as an individuals or groups behavior.

Ontologies can be used to model and capture the structure of formal semantics of social networks.

Wennerberg [15] describes how the structure of a network can be modeled using a semantic web ontology. Ontologies model, present and document the concepts and properties of a certain domain. Having the social nature of the networks as the domain of the study, ontologies can capture the concepts of relationships, individuals and their respective properties. Inference mechanism gives ontologies the ability of inferring new information using rules which could be of great importance in social context.

A set of existing efforts on modeling social network on the semantic web could be mentioned here.

Cantador et al. [17], model a social semantic network by utilizing ontology as a basis for clustering the user profiles in a social networking community. The ontology represents the domain of user's cognitive patterns, such as interests and preferences. Resulting ontological instances, take the shape of a semantic network of interrelated domain concepts and user profiles.

A similar effort [18] uses ontology at the core of a semantic web-enabled application. This ontology generates a social network of users and their interests. Generated ontological networks are used in order to detect and filter the Conflict of Interest (COI) relationships in an academic context, comprising authors and reviewers of papers.

In a similar effort with the same context, Mika [19] uses ontologies, in the context of a semantic web-driven application system and Flink [19], for modeling, capturing and visualizing the social network of researchers.

B. Trust overview

Being the key to any interaction procedure in human societies, trust has been the subject of studies to many fields of research and science such as sociology and psychology, as well as of course computer science.

Because of its importance and significance, trust has been harvested as a field of research in for example decentralized access control, public key certification, reputation systems for peer to peer networks, and mobile ad-hoc networks.

Despite the fact that there has been a variety of definitions for trust, there has not been an agreement on a generic definition of trust. Researchers mostly have defined trust, depending on the context and the orientation of the paper they have written or the experiments they have been conducting. As a matter of fact most of these definitions are specific to the context of the work being done.

Lack of consensus on generic trust definition makes us realize the importance of having a definition which is context-neutral and general enough to be applied to different fields of research and different contexts.

Trust is a complex issue, relating to fairness and straightforwardness, honesty and sincerity of a person or the service this person might offer.

Grandison [20] defines the trust in the following manner; "*Trust* is the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context". "Distrust may be a useful concept to specify as a means of revoking previously agreed trust or for environments when entities are trusted, by default, and it is necessary to identify some entities which are not trusted", according to [20].

Distrust is defined as "the lack of firm belief in the competence of an entity to act dependably, securely and reliably within a specified context" [20] [21].

1) Trust components, properties and sources

Trust is presented in the form a *relationship* between two parties. These two parties, often individuals or agents representing those individuals, are represented as *trustor* or *source*, which is defined to be the entity which seeks trust or trust related operations such as evaluation in other entity, *trustee* or *sink*, which is the entity that is trusted or it has been requested for trustworthiness-related evaluation. Trust is seen as having a *purpose* or a *context*. For instance, Alice trusts Bob as a doctor, but she might not trust Bob as a car mechanic, adopted from [20] [24]. In addition, a trust relation might also have a *trust metric*, which can be quantitative or qualitative, characterizing the degree to which the trustor trusts the trustee. This quality or quantity represents the *intensity* and *level* of trust. This quality and quantity can be evaluated by using an algorithm or *mechanism* which derives trust, according to the metric. For instance, Alice might trust

Bob as a doctor very much, while she only moderately trusts Martin as a doctor, adopted from [20] [24].

So far we have realized trust as a *computational value* depicted by a *relationship*, described inside a *specific context* and measured by a *metric* and is evaluated by a *mechanism*.

Some important properties of trust are stated and discussed [20] [24].

For instance, subjectivity (difference in judgments of two people on the same entity's trustworthiness) or transitivity (If transitive, when Alice trusts Bob and Bob trusts Cherry, Alice will trust Cherry, adopted from [20] [24]). One of the most important subjects of discussion on properties and components of trust is the difference being made between trust in performance and trust in recommendation [20] [24].

First, there is a difference between trust in an entity to perform an action (*trust in performance*), and trust in an entity to recommend other entities to perform that action (*trust in recommendation*). This is the distinction between Cherry trusting Bob as a dentist, and Cherry trusting Bob to recommend a good dentist, according to [20] [24].

Another difference is based on existence of recommenders. There is a difference between the trust that is directly observed by trustor from trustee and the trust that is conveyed and inferred from the recommenders' trust.

As a result, this difference can be sampled between Cherry trusting Bob as dentist, resulting from Cherry's own direct observation and evaluation from Bob, and Cherry trusting Bob as a dentist, based on the fact that she trusts Shawn as a recommender for a good doctor and on the fact that Shawn trusts (and perhaps recommends) Bob to be a good dentist, adopted from [20] [24].

During the observations made by [25], a set of sources of trust are identified, in both atomic (direct trust) and compound (social trust) forms.

Trust is the experience gained from an interaction between two individuals. So the actual experience is the source of trust. Considering the experience, or source of trust between two certain persons and individuals, this type of trust can be referred to as inter-individual trust or what is commonly referred to as *direct trust*, according to [25].

We can consider a setting of individuals across a web or network. If we consider this society of nodes and present the trust in this society and in this setting, then we are dealing with a new type of trust originating from the experiences gathered by a group of nodes or individuals. This new type of trust has its own source, from trust propagation in social settings or networks.

This type is called *relational trust, social networks driven trust* or in simple form, *social trust*, according to [25].

2) Trust computation in Web of Trust

Most of the models proposed for modeling trust on semantic web are more focused on probabilistic views of trust. They model trust using probabilities assigned as labels to the edges of the networks according to specified trust metric.

In order to derive and infer trust, edges are traversed and probabilistic trust values are gathered along the edges and using mechanism adopted, the trust value along the trust path will be computed and inferred. This setting is referred to as a Web of Trust. There are two reasons for making web of trust a candidate for adoption to trust in semantic web computation scenarios. First, both systems are open. Second, trust is considered as being transitive in both settings.

Web of Trust was a system that was introduced under the context of security and privacy systems, for instance PGP [55]. In this setting everyone can sign each other's key and act as certificate holder or certificate authority. Openness states the demand and need for metrics. Need for metrics, establishes and proves the relativity and computability of trust. The need for scalable trust metrics has been discussed and studied extensively [51] [52]. When metrics are applied all the links can carry them and trust can be inferred [27].

Under the assumption of trust transitivity and by enforcing metrics, pathways of trust can be formed and web of trust can be crawled and walked [53].

As stated, semantic web is a similar scenario in which each agent that forms a node on a network is connected to other nodes, agents, and these links and connections form a web of trust. In order to allow everyone, represented by an agent, to evaluate the statements of others in this open and heterogeneous environments, mechanisms and algorithms are developed or adopted to allow everyone to infer and evaluate trust in others using the trust metric-labeled links on the networks of trust.

3) Trust networks

The work in this field is mostly focused on the mathematical notion and presentation of networks but the amount of the practical work is limited.

Most of the works in this field do not consider design of larger infrastructures and ecosystems. Trust networks are described as weighted graph structures with directed edges. The edges in the generated graphs represent connections and relationships between individuals. Watts introduces the properties of a small world network [37]. He describes a model called ß-model [37] in order to model, construct and generate the structure of social systems. Many social systems have used this model within their infrastructure [34] [35] [36] [37] [38] [27].

Golbeck has done an extensive research effort on trust networks on semantic web, [27] [28] [29] [31]. She has constructed an ontology of trust, combining RDF and FOAF vocabulary to describe relationships comprising trust

networks. She has created applications on resulting networks of trust based on her ontology. These applications range from email filtering, TrustMail [27] [28], to web-based recommendation systems, FilmTrust [31].

Brondsema and Schamp [10] have created a system called Konfidi [33] that combines a trust network with the PGP Web-of-Trust (WOT). The system implements a metric and mechanism for inferring the trust on the networks formed. The generated network creates trust pathways in between email sender and receiver that can be crawled and using trust mechanism and metric, trust values are inferred [10].

III. EVALUATING TRUST ONTOLOGIES

This section gives an overview in some of the most important and influential works in modeling and designing trust ontologies. After giving a state-of-art overview in the observed ontologies, a framework for comparing and evaluating trust ontologies is introduced and the studied approaches are compared accordingly.

A. State-of-art in trust ontologies

As introduced earlier, Friend-Of-A-Friend (FOAF) [6] represents a vocabulary and introduces an ontology for describing a web of connected individuals.

This ontology can serve as a tool to model and eventually create a network of society of users by describing personal information about each person (realizing the node itself) and by describing personal information regarding a set of users whom the user knows about (realizing the neighbors on the network). Nodes on such a network are identified by their email address and email serves as their unique identification.

1) Golbeck's trust ontology

Jennifer Golbeck [27], introduces an ontology, that creates an important schema which extends FOAF by using *foaf:Person*, giving the users this possibility to state and represent their trust in individuals they know.

Metric used to express trust is a value on the scalar range of 0-9, in which each scale represents a trust level. These levels are set as properties under the domain of *foaf:Person*.

These levels correspond to: *Distrusts absolutely*, *Distrusts highly*, *Distrusts moderately*, *Distrusts slightly*, *Trusts neutrally*, *Trusts slightly*, *Trusts Moderately*, *Trusts highly*, *Trusts absolutely*, according to [27].

Context was introduced as a property of trust. Trust is context-sensitive, as a result meaning and semantics of trust can change depending on the context. This notion is represented in this ontology under *general trust* or *specific trust* or *topical trust*, according to [27].

For instance, Alice might trust Bob greatly on driving cars but might distrust Bob totally on repairing cars, adopted from [27]. In order to depict general trust within Golbeck's trust ontology, trust ratings (in the form of *trustsHighly* or *trustsModerately*) are described as properties in range of a person class under the range of another person.

To describe specific trust and topical trust, other sets of properties are introduced. These properties correspond to the nine values above, but are used to represent trust regarding a specific *topic* (for instance "*distrustsAbsolutelyRe*," "*trustsModeratelyRe*," etc), expressing the level of trust regarding a certain topic such as driving or dishwashing. The range of these properties is the "*trustsRegarding*", which has been defined to combine a person and a topic of trust. The "*trustsRegarding*" class has two properties: "*trustsPerson*" presenting the person being trusted (trustor), and "*trustsOnSubject*" presenting the subject that trust is stated towards, according to [27].

By having this ability we can query for trust about a person on a specific subject and it is possible also to infer trust on result trust network along the edges where given topic creates the connection and we can crawl along these paths to infer the trust value eventually.

2) Toivonen and Denker's Message and Context Ontology

Toivonen and Denker [41], study the trust in the context of communication and messaging. They state that there are many factors which can have immense impact on the honesty and trustworthiness of the messages we send and receive. The context-sensitivity of trust has been realized and taken into account in their work.

The work focuses on drastic changes that many issues, namely reputation, credibility, reliability, trustworthiness and honesty could have, and how they affect the progress of establishing and grounding trust, according to [41].

As a result of the work being done, a set of ontologies have been defined to capture context-sensitive messaging and trust. An ontology is developed to capture and denote the role of context-related properties and information. This ontology captures the domain of message communication and exchange and describes how the context information is actually attached to the messages. This ontology is constructed mainly to visualize how trust is related to message and communication.

It is important to note that this ontology extends the topical trust ontology of Golbeck [27], introduced earlier, and it relates the notion of trust to communication and messaging context. Basic idea behind this extension is that:" The topic of a message can have impact on its trust level" [27].

As a result, this trust ontology could be seen as an extension to topical trust ontology realizing the fact how trust can be fused within messages exchanged in the context of a communication environment. This concept is modeled and presented using *trustsRegarding* property. Links and connections between

persons are modeled by the *Trusts* property. Sub-properties of these two relationships conform to trust levels of Golbeck's ontology [27].

In order to model the relation of trust to the context, the *ctxTRUSTS* property is used. If we consider the environment of a simple communication setting, we see the sender, receiver and the communication network mediating them. The messages exchanged in between parties always have contexts, attached to them which in turn allow the computation of *ctxTRUSTS* properties through *Trusts* and *trustsRegarding* properties, according to [41].

3) Proof Markup Language's trust Ontology

Inference web [42] at Stanford University, has built a semantic web-enabled knowledge platform and infrastructure. This platform is designated to help users on the network to exploit the value of semantic web technologies in order to give and get trust ratings to and from resources on the web. This process is referred to as justification of resources. To this end, a language called PML is used.

PML [26] (Proof Markup Language) contains a term set for encoding the justifications and is designated to work in a question answering fashion [44]. PML is designated to help software agents to filter the resources on the web of semantics by proof checking them and justifying the credibility of these resources, on behalf of the users.

PML ontology contains three sub-ontologies including: provenance ontology, justification ontology and most importantly trust ontology which captures honesty and trustworthiness statements pertaining to resources.

The trust ontology [26] is one of the most important components in PML ontology and we briefly describe the structure of this ontology.

The approach presented here is modeling close notions of trust and belief and how it affects the credibility of resources on the web.

Notions of belief and trust, with respect to their close semantics, have been presented closely in this ontology. Ontology structure presents the trust and belief relations between a source and a sink (which are both realized and presented using agents) with respect to information from document source under investigation by respective agents.

The belief relation shows the belief of an agent about the source. The specific belief has a status (e.g. believes, disbelieves, ignorant). The trust relation shows an agent's overall beliefs about information from the specified source. The metric defined for trust and belief is probabilistic and for both elements a value between range of 0 and 1 has been designated.

4) Konfidi's trust ontology

With respect to metrics used for presenting the trust computational values and modeling the mathematical notion of trust, there exist two approaches: presenting a trust metric with discrete values and metrics with continuous values. Brondsema and Schamp [10] model and represent trust and distrust in a similar fashion using continuous values. Having continuous range of values allows easier propagation of trust values, along the edges on the networks, using inference mechanisms.

They represent the relationship as the class and main concept of the ontology. Each relation is directed from source (truster) to sink (trustee). Properties of relations are wrapped under the concept of trust item. The most important feature of this work is, like Jennifer Golbeck's ontology [27], they have incorporated the notion of "Topical trust" in their ontology. It is used as an attribute and property, which allows to state different features and properties of a relationship. Trust topics and trust values are stated as properties of the trust relationship.

In order to describe trust relationships, an ontology is presented using RDF, which in turn eases extending the FOAF vocabulary and profiles. Using the RDF properties, and taking into account that relationship can be described using FOAF vocabulary and ontology, then trust relationships can be described using trust ontology. Other technology that has been integrated is WOT [45] [46] (web-of-trust), that is used to describe web-of-trust resources such as key fingerprints, signature and signing capabilities and identity assurance [10] [46]. Ontology's RDF schema is made of 2 classes or concepts and 5 attributes or properties. As mentioned, the primary concept is Relationship between two people. Like most trust ontologies, there are two properties that are required for every Relationship, and they form the endpoints of every relationship; truster and trusted using FOAF vocabulary, both truster and trusted have *foaf:Person* objects as their targets.

Using WOT vocabulary, FOAF-defined Persons should also contain at least one *wot:fingerprint* property specifying the PGP, web-of-trust fingerprint of a public key held by the individual the Person refers to. Most importantly, this property serves for two reasons; first assures the identity of these people described on the both ends of relationship, and it also says if one of the people does not hold any keys then system can ignore instantiating a relationship between them.

B. Comparison and analysis

In this section we will compare some of the most important afore mentioned ontologies. We will try to point out common and shared points between mentioned ontologies, and we will also try to address strong and weak points among them. Table1 compares the ontologies reviewed so far based on the components of the ontologies.

TABLE I
COMPARISON AMONG TRUST ONTOLOGIES BASED ON ONTOLOGY COMPONENT STRUCTURE

Trust Ontologies	*Concept(s)*	*Relationship(s)*	*Instance(s)*	*Axiom(s)*
Jennifer Golbeck	Topical trust, Agent, Person	trustRegarding *(between agent and Topical trust)*	trust0...trust10 (range of trust metric), trustSubject, trustValue, trustedAgent, trustedPerson (subproperty of trustedAgent), trustRegarding	"A *Person or Agent* (e.g. Alice) *trustsHighlyRe* (*trust10*) *trustRegarding* a *trustedPerson or trustedAgent* (e.g. Bob) On *trustSubject* (e.g. Driving)"
Toivonen, Denker	Person, Topic, Receiver, Message	Trusts *(between Persons)*, ctxTRUSTS *(between receiver and message)*, trustsRegarding *(between Person and Topic)*	trustRegarding, reTopic, [trustsAboslutelyRe … distrustsAbsolutelyRe], ctxTRUSTS, [ctxtrustsAbosolutely … ctxdistrustsAboslutely], trustsRegarding, Trusts, rePerson, [trustsAboslutely … distrustsAboslutely]	Multiple axioms are inferable, for instance; 1) Stating topical trust; "A *Person* (Alice) *trustsAboslutelyRe trustsRegarding* (relationship) the *Topic* (Driving)", 2) Stating trust between two persons; "a *Person* (Alice) *trusts* another *Person* (Bob) *trustsAboslutely*"
PML	Belief Element, Trust Element, FloatMetric	Belief Relation (using *hasBelievedInformation* and *hasBelievingAgent* between *Agent*, *information* and *source*), Trust Relation (using *hasTrustee* and *hasTrustor* between *Agent*, *information* and *source*)	Agent, Source, Information, hasBelievedInformation, hasBelievingAgent, hasTrustee, hasTrustor, hasFloatValue,	Two kinds of Axioms regarding the trust and belief of agent in an information from a source can be inferred, for instance; 2) Stating trust; "*FloatTrust*, *hasTrustee* and *hasTrustor* *(agent: user's browser)* And *hasFloatValue* with *FloatMetric* (0.55). "
Konfidi	Relationship Item	About *(Between Item and Relationship)*	About, Truster, Trusted, Rating, Topic,	Trust Relationships can be stated like the following axiom; "A (trust) *Relationship* between *truster* (Alice) and *trusted* (Bob) exists, which is *about* trust *topic* (Cooking) with trust *rating* (.95)."

To further analyze the study we have done so far let's consider a set of analysis subjects that affect the discussion on the comparison between ontologies.

Depending on the context and the subject of the study certain approaches are used and implemented. If the subject of study is considering ontologies for knowledge management then, it is preferred to use an algorithm to compare ontologies, since such ontologies may be heavy and may contain a large number of concepts and properties. As a matter of fact we can use *weight* of ontology as the basis of comparison.

As all trust ontologies convey the same meaning and that is representation and modeling trust relationship, *Context* seems to be an important issue. So, we can compare trust ontologies depending on the context they have been modeled and considered in.

Since a model should also ease and facilitate the *inference* and computation of trust, then inference should be also an important topic to consider while analyzing trust ontologies.

Trust ontologies are used to generate trust networks and they serve as the gear to rotate the automation of trust network generation, inference and maintenance, therefore we can consider comparing ontologies based on the *ease of implementation* as well.

Ontologies should allow expressivity of trust statements. As axioms represent the trust expressions and statements on the social community of trust, then we can also consider the *semantic expressivity* of the axioms inferred based on the respective trust ontologies. Semantics of trust should be easy to understand and should allow inference and justifications.

The more trust ontologies incorporate and integrate technologies and *vocabularies* that create expressive and referenced, the more they will be easy to implement. Importing technologies and vocabularies make ontologies rich. As a matter of fact we can also consider basing our justification based on the number or technologies used in an ontology.

1) Weight

Considering the size of ontologies, Konfidi is the lightest ontology by having only two main concepts and 5 properties and only one single relationship.

PML has 5 main concepts, but there are 2 types of relationship existing with 8 instances, making PML trust ontology the second in the place.

While Golbeck's ontology has one single main concept (topical trust) and two other derived concepts (person and agent), 16 properties and one relationship, making it the third place holder. Trust ontology of Denker/Toivonen has 4 main concepts and 3 types of relationships, making it the heaviest ontological representation of all.

The reason for the excessive size of the number of properties of Golbeck and Denker ontologies, is the trust metric used; if the discrete scale between 0 to 10 was not chosen, and a probabilistic approach was used then the mentioned ontologies would be way lighter, bringing the total number of elements to 11 in Golbeck and to 14 in Denker/Toivonen, make them the top place holders at first and second place.

As a matter of fact we can conclude here that the choice of trust metric and the approach toward computational aspect of trust measurement could affect the size of ontology drastically.

2) Context / domain dependence

As described context is one of the most important subjects to consider while building a trust model for a domain of study. We also have to consider that there are main elements that affect the construction of trust ontologies that could alter their structure.

We want to consider construction of an ontology that could be based on the main axes of trust, semantic web and social network. Considering the main axes and elements that affect the structure of ontology, could create a drastically different ontology with a set of different components.

For instance if we consider the trust in service-oriented environments, we have to consider trust as a notion close to security, rather than belief and judgment. In that context trust is more close to reputation, while trust in the context of semantic web and semantic web driven social communities is more close to belief and justification.

As a result, context has a considerable impact of the constructing elements of trust ontologies.

Among the ontologies considered, Denker/Toivonen is the most context-dependant ontology, as the context of the trust study is communication and message-exchange. Taking a look into trust concepts incorporated into this ontology, we realize that the notion of trust relationship is tangled up in communicational concepts (Communication network, Message) make it completely dependent to communication context although the rest of the trust components are very well-engineered.

Since the trust ontology of PML is an axis of a triangle of provenance, justification and trust ontology, all of the mentioned ontologies are incorporated and imported into each other to take advantage of the technical facilities of ontologies description and consumption. This feature makes trust axis of PML ontology, dependent to other three ontologies and incorporating such ontology demands incorporation of the other two ontologies. At the same time this ontology is dedicated to evaluate and express the trust and belief of an agent into a piece of information taken from a source of information on the web. This feature makes it hard to express and conclude the trust between a set of persons, since the other pair should be described by agent as well, but it makes it easy to derive and justify the statements of a person and state the

belief and trust in the statement made by a person (for example on a social network). In general, the approach that PML follows is "Trust for Question Answering" [47]. As a result, PML trust ontology seems less context dependant in comparison to Denker/Toivonen and more customizable to the need for modeling trust, in general.

While Konfidi makes representation of trust in the context of social semantic networks fairly easy and straightforward, at the same time it is extensible and useful to different contexts and the future needs. Using the Konfidi's ontology, you can state a statement of topical trust between any set of resources or nodes (described by URI) on a semantic social network.

Golbeck's ontology seems the most essential and fundamental work on describing and stating trust using ontological modeling and representation for the consumption on semantic web. Both Konfidi and Golbeck's ontologies are among the most context and domain independent ontologies and that makes them easy to be customized and implemented in other domains of interests, demanding for modeling of trust.

We can state that the more ontology has components that directly expresses the trust relationships and has less components and properties related to other domains, the more context-independent it will be.

3) Inference capability

One of the most important issues while considering capturing of a domain inside the structure of ontology, is the reasoning based on that ontology.

Considering the subject of discussion, it should be possible to infer trust values easily using the corresponding trust ontology. As described, choice of trust metric plays a crucial role in the design and composition of ontology. Given a set of entities (for instance two persons located on a network), ontology should facilitate the inference of computational trust value for the given entities. There are certain factors that affect the efficient inference based on ontology such as the complexity and size of trust network generated. The lesser trust network generated is complex the lesser the inference mechanism implemented needs to be complex.

Golbeck's ontology was used for generating a network of semantic data, and was also used within a semantic web social network. Research has shown great inference capability for this ontology [27][28][29][31].Golbeck has studied the inference mechanisms and has created and implemented inference algorithm to study the trust inference based upon her trust ontology on two sets of trust networks, one a website for movie ratings and recommendations [30] and the other for spam filtering [28]. This makes Golbeck's trust ontology the only ontology widely used, implemented and inferred upon.

Konfidi is also tested against network of semantic data, and has shown good performance. Konfidi uses trust strategies to implement different sorts of inference mechanisms and algorithms, in order to test the inference capability of trust ontology, according to [10].

The inference capability of PML is implemented and has proven to be very effective as it is designated toward automatic resource evaluation.

It is important to consider that trust inference capability is an important factor that affects the implementation aspects of trust representation.

4) Semantic expressivity

Axioms that are inferred from trust ontologies express the semantics of trust. The more clear and expressive these axioms become the easier they will describe the semantics of trust within the implemented and stated context.

Golbeck's and Konfidi's respective ontologies state the semantic trust relationships very easy to understand and very expressive; for example using Konfidi; "*Relationship between truster (Alice) and trusted (Bob) exists, which is about trust topic (Cooking) with trust rating (.95).*" and using the Golbeck's ontology;" *"A Person (Alice) trustsHighlyRe trustRegarding a trustedPerson (Bob) on trustSubject (Driving)"*, adopted respectively from [27] [10].

As Denker/Toivonen use Golbeck's approach, but the axioms generated are less expressive as multiple contexts are taken under consideration and final driven axioms should have the notions of context, trust, communication. Considering all intermediary relationships for example, a trust relationship between person and topic could be described as; *"a Person (Alice) rePerson trustsAboslutelyRe (trust metric) trustsRegarding (relationship) reTopic Topic (Driving)"*, adopted from [27], which shows less expressivity than previous axioms.

As described in the table, PML has the less expressivity among all, but this is traded off with the inference capability of the ontology, as the inference should be consumed by software agents.

There seems to be a tradeoff between the expressivity of inference capabilities of ontologies; as the ontology becomes consumable by software agents, the less expressive the inference products become.

5) Size of trust networks

We discussed that the trust network should be automatically generated during runtime so we can analyze and evaluate and finally infer and compute the trust values based on the generated network.

As the size of the corresponding networks grows, the harder the crawling and walking the trust paths becomes. So, it is important to consider that the network generated could be analyzable and inferable. This has to do directly with the

structure based on which trust concepts and properties are presented and described.

For example Konfidi describes the topic and rating, as the extra edges on the network tree. The more topics we incorporate the larger the depth of the network generated becomes, so in order to increase the efficiency, authors of Konfidi's trust ontology state that the extra information attached to edges could be saved separately, according to [10].

As the semantic concept of trust relationship has been described very efficiently (using a small set of necessary elements, e.g. only one main concept), the networks generated are very well-formed. It is logical to state that the efficient design of ontology directly results on the efficient design of the networks generated and used.

As our ontology is introduced in next section, we use network size prospect to analyze the networks generated using our own ontology.

6) Vocabularies incorporated

As mentioned before, Golbeck's trust ontology was indeed a milestone in the field of the work being done for representing trust and belief in statements done on a semantic web-driven community and society.

She not only introduced a method of representing trust on semantic web and semantic web-powered societies, but she also introduced the notion of topical trust and subjective trust. By enabling the subjective trust we can state and represent how a sink and a source trust each other based on a specific subject and then measure this trust in subject and topic according to a specific trust metric.

Most of other works within trust representation on semantic web and semantic web-driven social networks either base their trust model completely or partially on Golbeck's trust ontology.

Denker and Toivonen incorporate the subjective and topical trust as well into their ontology. They also use the trust range of Golbeck for contextual trust and personal trust representation.

Konfidi also incorporates the topical trust. Although not standardized, topical or subjective trust is a requirement for any kind of model capturing the trust relationships. All of the studied ontologies take advantage of friend-of-a-friend (foaf) vocabulary. Golbeck and Konfidi use the foaf vocabulary to describe the two sides of trust relationship.

PML uses it to describe the agent that assesses and evaluates the information. Among studied ontologies, Konfidi incorporates and integrates the most number of vocabularies and technologies. In addition to foaf and topical trust vocabularies, Konfidi also incorporates relationship vocabulary [48] and it also uses WOT [45] (web of trust) vocabulary. Using the relationship vocabulary leaves space for adding other new features of trust relationships when needed; such as the date of initiation of trust relationship, terms of relationship, etc. Integrating different vocabularies, enriches the structure of the ontology, reduces the number of ontology components and eases the inference based upon the respective ontology.

Considering standardized vocabularies and ontologies, not only reduces the number of elements, but also eases future adoption of new properties of implemented vocabulary-driven features.

IV. Engineering and Construction of Trust Ontology

The same as all engineering sciences, in order to engineer an artifact, an iterative process should be considered where each step proliferates and extends the previous step in the loop to construct the artifact under focus.

Ontology engineering and learning is a semi-automatic process, consisting of six main interrelated phases, according to [50] [49].

These phases include: domain understanding, data understanding, task definition, ontology learning, ontology evaluation and refinement with human in the loop, respectively taken from [50] [49].

We use this approach in order to construct and build our trust ontology. We can state that our experience not only can serve as a methodology and mechanism for ontology construction but also, considering the domain of our problem, it can serve as a guide to engineering and construction of trust representations and protocols using ontologies.

1) Determining the domain and scope

Considering the domain of problem, we are engineering an ontology, which serves as the representational structure of the relationship visualizing trust and trustworthiness of a set of individuals based on a social network.

We can state that this ontology rotates on four main axes; Trust, Relation, Social network, Semantic Web. So, we can state that the domain of our ontology is representation of trust within a social network based on semantic web.

2) Understanding and learning the data

Domain and scope of ontology create boundary that captures the data relevant to the ontology under consideration.

Since our ontology serves as a representational model, then we understand that the focus will be put on the data that are represented, and that is trust relationships.

As relationships are compound data made of couple of atomic subcomponents, then atoms of relationships will form the data of our ontological domain. Relationships are described

between entities and these entities are individuals on the social network, connected with trust relationship together.

We consider persons on the social network, so data about people will serve as our data. Data about people on the social semantic network are described within FOAF files, which are described using RDF. At the same time Web-of-trust also provides data about the identity of people on our network as well as availability of links between these people on the network of relationships. As such relationships should describe trust as well, properties of trust are also among the pieces of information that are also useful to create data for ontology, such as the measurement and metric value used to describe the value of trustworthiness. In order to be able to describe the subject of the trustworthiness evaluation, we need also a subject list, so, available subjects and topics can be mentioned as available data on the domain.

The data needed within this ontology is information that composes trust relationships and properties that relationships have. The main data would be people's relationships and properties describing them and their relationships; here as mentioned metadata FOAF profiles of people can compose such relationships.

3) Defining tasks

Available data describe not only information about people that make the atoms of relationship molecules but also the properties of relationships. When domain is specified and the data available are recognized and learned, usages and functionalities of ontology being constructed is specified. Taking into consideration the domain and scope of ontology, which is representation of trust relationships and the data available that are information about people creating the relationships, we can state that the task of such ontology would be clearly, describing and representing trust relationships.

4) Ontology learning

Using the knowledge acquisition and learning capabilities with the help of our construction and development environment, we are able to learn the ontology.

As the main component of the ontology is relationship, that represents the connection between entities on the network then *relationship* itself serves as the main component and concept of this ontology.

We can think of relationship as a composite object made of subcomponents that reside within the relationship and describe the properties of relationship. Each relationship describes an edge on the network, this edge exists between a set of nodes. These nodes represent starts and ends of directed edges (of relationship). *hasTruster* and *hasTrustee* respectively represent the two important properties of relationship on a network. So far we have learned the elements of relationship on a social network.

Every relationship has a set of main properties, which describe the nature and purpose of relationship. These properties specify the details of trust relation. Each trust relationship has a *topic* or *subject* (topical or subjective trust). In order to make trust computable, on any existing edge on the network there should be a value. This value represents the trust metric used for the representation of trust relationship. So, we can consider Value also as a main property of relationship.

Now that we have learned the main elements of ontology, it appears most of the trust ontologies share the same components described so far. Relationship described using ontologies have a set of *auxiliary properties*, as well. Using this component we can put more details on the relationship and we can give it more weight and mass. It is important to realize that only properties that have less importance than main properties, are described using these properties. These properties are used to give more weight to Relationship. Using a separate element for auxiliary properties leaves space for future extensions that are needed to add to the network

Trust relationships are context-sensitive. *Context* describes whether this relationship is described inside a personal network or a business network. By using context, we can make networks of different types. Using this element we can create simple networks and hybrid networks.

For instance, simple networks are either a personal network (such as Orkut [56]) or a business network (such as LinkedIn [57]). We can have a relationship in the context of a personal network. We can also have a simple trust relationship in the context of a business network or perhaps a business environment. When the source is from a personal network and is connected to a sink from a business network we have connected two networks of simple type, creating a hybrid network. As a matter of fact context type can give more details about the type of network where this relationship is described in. This auxiliary element gives more details about the type of network the relationship is based upon.

Considering the reason that a relationship can be established based upon, we have also incorporated a *Goal* property describing the reason that a relationship was based upon. A relationship can have a goal that describes why respective relationship is formed. For instance on a social network, usually the goal for establishing a relationship is friendship, or on a business network, it is seeking business partnership.

The most important subsidiary and optional property that we have considered in our ontology is having a *recommender* as the initiator of the trust relationship establishment. *hasRecommender* is an auxiliary property describing a person on the network that has recommended trustee, or the sink of relationship, to the truster. In other words, we have described notion of "trust in recommendation" in order to shape and form a relationship, initiated from truster, ended up in trustee, based on guarantee of trust recommender. Using such property we can create networks of different strengths; we can have networks of weak links and strong links.

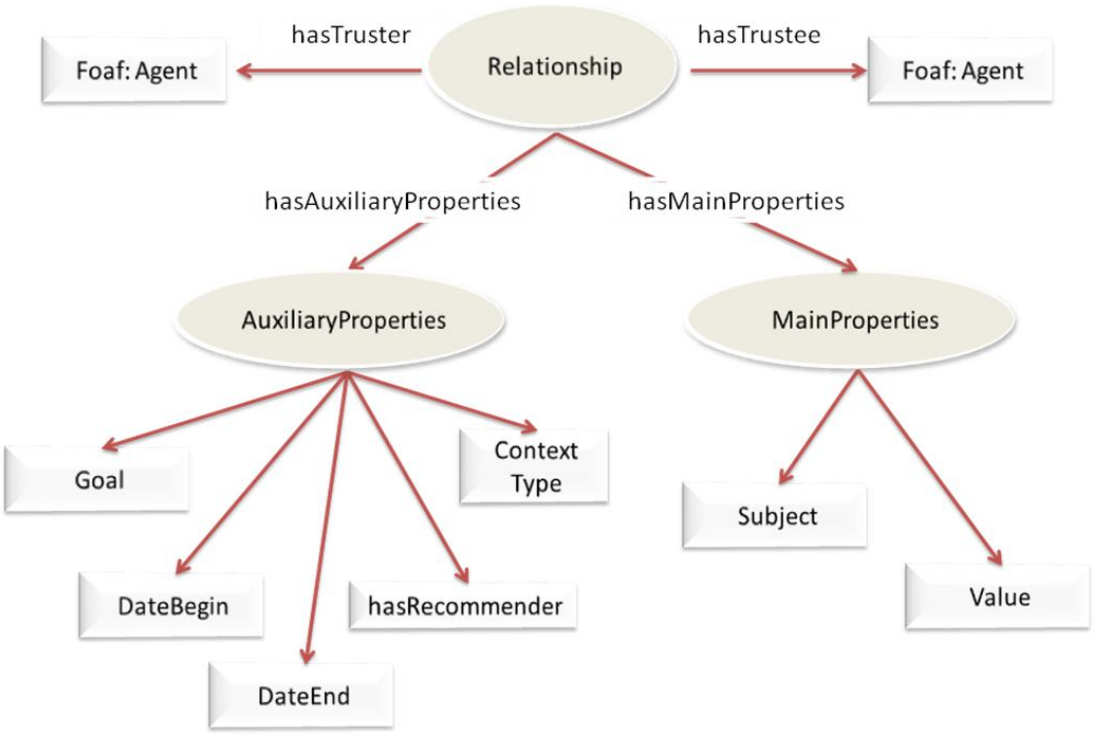


Fig. 1. Structure of our trust ontology, 3 main concepts of trust ontology as well as two edges connecting them together. [2]

A *strong link* is a relationship based upon the recommendation of an entity. The more recommenders a relationship has, the stronger this relationship become. A *weak link* is a relationship that has no recommenders.

When speaking in terms of trust, in context of information systems, a way of achieving trust is using a recommender. Considering the transitivity property of trust, the trust in recommender is used in certificate authorities to achieve trust in a third-party. We can take advantage of this property in semantic-web driven social networks to create strong paths in order to use them as the path for aggregating and computing trust values along the network.

By specifying auxiliary properties we follow two important goals; Adding more details about relations to ontology and giving more meaning and details to specification of relationships, as well as leaving space for adding more elements describing the other aspects of relationships that may be needed in future.

5) Ontology evaluation

As ontology development is a semi-automatic approach and demands involvement of both human and machines, in this phase as well as previous phase we take advantage of using an automated tool in order to build and evaluate our ontology. In this phase we build and evaluate the ontology learned in previous phase. By evaluating the ontology, we estimate the quality of the modeled solution to the addressed tasks defined in previous sections.

It is worth mentioning that in most of the phases of ontology engineering the role of human wouldn't be completely fade and human will participate in almost all of the phases of ontology development. In order to model and describe the elements and components of the ontology we use Protégé[3] ontology editor and knowledge acquisition system.

Figure 1 visualizes the structure of our trust ontology.

As shown our ontology has 3 main concepts or classes that capture the structure of the trust relationships on the networks.

Relationship is the main element and concept of our ontology. *MainProperties* and *AuxiliaryProperties* are the other main components of our ontology. We have two associations that connect both *MainProperties* and *AuxiliaryProperties* to *Relationship*. These associations are *hasMainProperties* and *hasAuxiliaryProperties*.

Relationship always has a sink and a source, which we have described here as *truster* and *trustee*. Both *hasTruster* and *hasTrustee* are defined on the range of *foaf:Agent* which enables us to describe relationships in the context of semantic social ecosystems. This agent can be a person, an organization or just a software agent. Each Relationship has to have a truster and a trustee and at least one main property. Without these mentioned elements, a relationship is partial and partial relations are undefined using our ontology. In order to ensure having at least these mentioned elements, we have put restrictions on ontology subcomponents. Restriction defines a blank node with restrictions. It refers to the property that is constrained and defines the restriction itself. Cardinality

[3] *Protégé*, http://protege.stanford.edu/

constraints define how many times the property can be used on an instance of a class. We have minimum, maximum, exact cardinalities.

We have used two exact cardinalities on *hasTrustee* and *hasTruster*, in order to state having exactly one truster and one trustee for a relationship. We have also used minimum cardinality for *hasMainProperties* to make sure having at least a topic and a value for each relationship, and since we can have more than one topic to base the relation upon, we have used minimum cardinality (at least).

MainProperties element has two main properties; *Subject* and *Value*. We have described these two properties using data type properties, in OWL (Web Ontology Language). Subject takes string value. It is recommended that subject taxonomies or topic ontologies be defined, so we can use a common namespace for describing topics and subjects. Each relationship can have multiple main properties, which means it can be about different topics and subjects, but each main property has to have one and only one topic and only one value.

For instance in the relationship between Alice and Bob, Alice can completely trust Bob on Driving (Subject="Driving", Value="0.95"), and also can distrust Bob on Cooking completely (Subject="Cooking", Value="0.10"). This constitutes two distinct main properties in relationship between Alice and Bob. But we cannot have multiple subjects and values in the *MainProperties* of Alice and Bob on Cooking, for example. In order to enforce this property we have put restriction on both properties of value and subject. By using exact cardinality restriction we have enforced having exactly one subject and exactly one value for each item of trust within a relationship.

Finally, *AuxiliaryProperties* concept of domain has 5 properties and also leaves space for more properties whenever needed. *AuxiliaryProperties* has an object property and 4 data type properties. It has *hasRecommender*, which is the element describing the strength of relationship and is defined on the range of *foaf:agent* that lets us to state which node on the network is the recommender for the establishment of this relationship. *ContextType* is defined as a string data type property that states the context of the trust network, the relationship is based on. *Goal* of the relationship is also defined using a string data type property. *DateBegin* and *DateEnd* are described using Date data-type property. Clearly we don't need to have restrictions on any single property of *AuxiliaryProperties* concept.

6) Discussion

As modeling trust is the main target of our work, a brief discussion on the notion of trust and how we have modeled the trust in our approach seems necessary.

As discussed, trust is a context-sensitive issue. While considering the context of the trust ontology and trust analysis, we realize that this context is a multi-dimensional entity composed of two substantial and main dimensions; semantic web and social networks. Trust in the domain of social semantic networks, has three relatively close notions such as belief, provenance and justification.

Some of these notions have very close and sometimes overlapping meaning to trust. Among mentioned notions, belief seems to be a very close notion to trust. It seems that belief and trust go hand in hand.

Discussion on modeling belief has a long background. The work on belief goes back to Willard Van Orman Quine's "web of belief" [22]. A reminiscent of web of trust is created by [23] and is weaved into semantic web. They define web of belief as following "by cognitively viewing knowledge as individuals' rational beliefs about the world, individuals share knowledge and form a distributed knowledge network, which is called the *web of belief*, where rational belief links individuals with world facts and trust interlinks individuals as external information sources." [23].

In our work, we have only considered modeling trust and distrust. Considering modeling other notions described takes a great effort and deal of modeling, as each one of these mentioned notions demand their own properties and eventually their own ontologies.

As a matter of fact, as we have generalized the notion of trust relationship in our approach to Relationship, then we have provided enough space for future extension. We can build belief ontology that can be imported within our trust ontology and certain elements of these ontologies can be shared and consumed whenever needed. Aside from such possibility then there is a need for future research for defining the nature, usage and representation of belief and judgment in semantic social networks.

Using our ontology, we can describe trust in other people on the network regarding a certain topic. Taking into account the discussions we had in previous section, what we are describing here is trust in performance.

When we state that "Alice trusts Bob regarding Driving", this means that, "Alice trusts in *eventuality of performance* of Bob to some extent, when the act of driving is performed". Trust in performance describes that truster states the trust in the performance of act of trustee, when this act is performed. This trust uses a probabilistic approach to describe trust relationships, so we can say how much someone trusts the other on a range between 0 and 1.

For example, as shown previously we can state, Alice trusts Bob completely regarding a topic. This amount of trust is mapped to a floating point value between 0 and 1, so we can state range of 0.9 to 0.99, is a range showing that you completely trust the person you are expressing trustworthiness about. Considering the discrete range of Golbeck's ontology, which is between trust0 to trust10, then we realize that we are having an implicit mapping from a range of discrete values to

a range of concrete values. Choice of trust topic is also considerable for improvement in future works.

As we stated, we have modeled *Specific Trust* and we have clearly eliminated the notion of general trust. It is important to point out that a relationship should have at least a topic. One of the important notions that we can consider discussing here, is *distrust*.

For instance, "*Alice distrusts Bob regarding babysitting to some extent (0.65)"*, using our ontology it can be also stated like "*Alice trusts Bob regarding babysitting to some (complementary) extent (0.35)*", adopted from [27] [10]. As it is clear we have modeled distrust, implicitly. We have assumed that there is a tradeoff between trust and distrust on the same topic.

We can also model feelings using our trust model. If we take all of the evaluation values for a relation, and average it, we can derive the amount of feelings between the trustee and truster. We can derive negative or positive feelings. If there are certain number of trust items (or *MainProperties*; subjects and values) for a relationship, for instance at least 3, we can consider taking average of the values and deriving a general feeling of truster for trustee.

For instance, if Alice has low trust values for Bob in all of the subjects in their relationships, then we can state that she has negative feelings for him, or vice versa. Although, there are many certain properties that should be considered that affect feelings of people for each other and trust is only one of them. Therefore, we can state here that more elements are needed to give us this ability to create feelings statements in our ontology.

We want to be able to choose two nodes, a source or truster and a sink or a trustee (trusted), and gather trust values on a path between them on the network and eventually compute a value representing the trust of truster in trustee. In order to address this problem; we have made sure that each relationships on the network has a value, and we have introduced recommenders.

Our ontology ensures that if there is a relationship (a link on the network) between two nodes, then this link has a value, although this value doesn't reflect the general trust value of trust between truster and trustee. In addition to using recommenders, we can use our ontology to create a network of recommendation on the network of trust.

We can use recommended links for our trust inference. As we described, recommendation can state the strength of an existing link, so we can use such "recommended link" for our inference along the paths. Theoretically, such paths are stronger and can give better values than other paths that do not have recommenders.

One of the main challenges in this context is dealing with distrust values, when encountered on the network. Values of distrust drop the aggregated values along the paths on the network, and there is no certain procedure or methodology to address dealing with this problem.

V. TRUST NETWORK ANALYSIS

We begin by analyzing a network of small size. This gives us the ability to easily, visualize and realize the structure of modeled relationships. Then we move to networks of larger size where we introduce two types of trust network structures; hybrid and meshed networks.

A. A small size network

Let us begin with the smallest network size, possible; a network of two people, with a single relationship, containing a main property and an auxiliary property. Let us consider modeling following relational semantics for this atomic network:

"Alice trusts bob in driving a lot."

Using our OWL trust schema and ontology, this network will be presented in RDF format as following;

```
<foaf:Person rdf: ID="Alice"/>
<foaf:Person rdf:ID="Bob"/>
<Relationship rdf:ID="Relationship_Alice_Bob">
      <hasTrustee rdf:resource="#Bob"/>
      <hasTruster rdf:resource="#Alice"/>
   <hasMainProperties>
      <MainProperties rdf:ID="MainProperties_Alice_Bob">
         <Subject rdf:datatype="&xsd;string">Driving</Subject>
         <Value rdf:datatype="&xsd;float">0.95</Value>
      </MainProperties>
   </hasMainProperties>
</Relationship>
```

B. Hybrid trust networks

Here, we will consider 2 groups of people, representing two networks of different contexts. Each group of four people is interrelated and interlinked, forming a *simple network*. At the same time a set of these people are connected outside of their own local networks, to other foreign network.

These relations work as glue connecting networks of different context, creating *Hybrid networks.*

In hybrid network depicted in Figure 3, people located on one network, are shaping a personal context and their goals are more or less establishing friendship relations, while people on the other network are members of a business network, and their goals are establishing business partnerships and relationships and they could be colleagues in an office environment. It is also considerable to think of the business network as a business-value adding network, or a service oriented environment. In that case, then four latter members

can be software agents, which can also be described using our ontology.

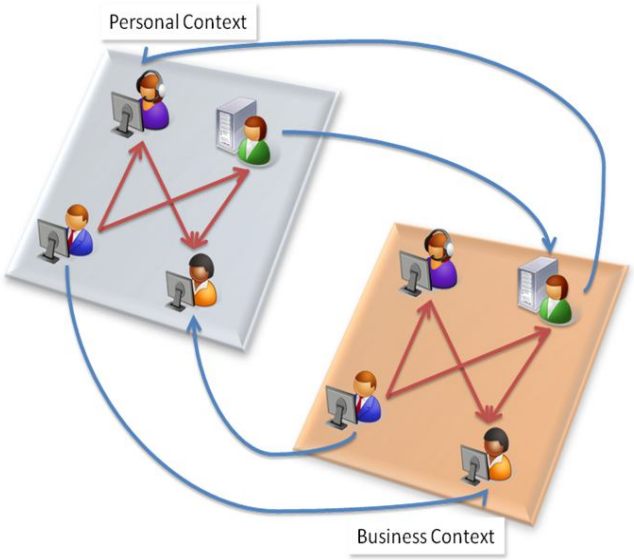


Fig.3. A hybrid network. Two connected networks of different contexts; a personal and a business network. Hybrid networks, contains 8 people and 12 relations. 8 links are interconnections (local), and 4 links are acting as glue connecting two networks (foreign).

In order to consider the structure and size of the network generated, a circular representation of network is given in Figure 4.

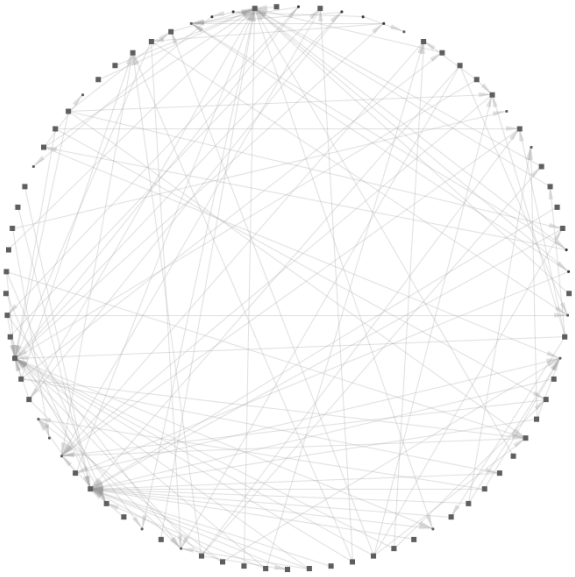


Fig.4. a circular representation of hybrid network subject to study. (Network contains 48 nodes and 92 edges)

Figure 4 visualizes the RDF trust network depicted in Figure 3. Figure 4 is visualized using Welkin[4].

C. *Meshed trust networks*

The motivation for studying larger networks of trust, was considering real-world scenarios of network formations. Such networks are complex, combined networks of different sizes and different contexts. We call these networks, *Meshed networks*.

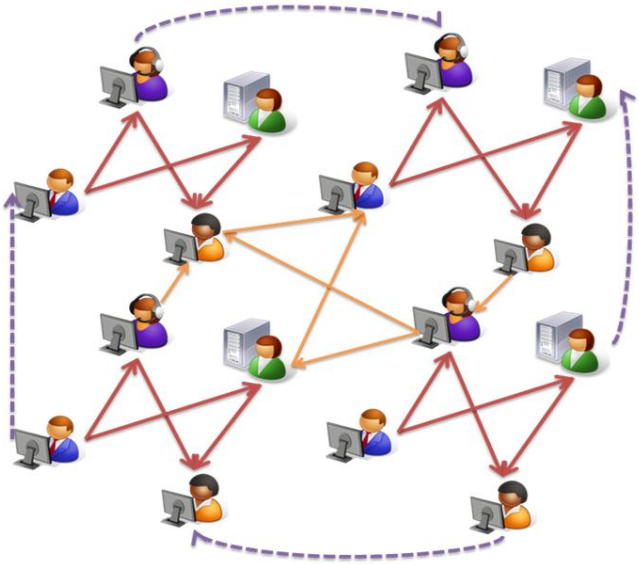


Fig. 5. A partial meshed network made-up of two connected hybrid networks. This network contains 16 people and 26 relations.

Meshed networks are considered networks, where every node is connected to all other nodes on the network.

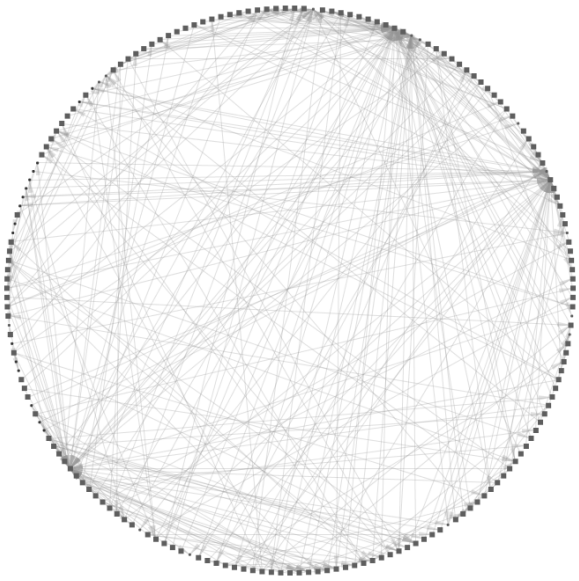


Fig. 6. A circular representation of meshed partial network. (Network contains 98 nodes and 198 edges)

As such, assumption is unrealistic, and there is only a subset of nodes available that are fully connected to all other nodes, we consider *partial* and *fully connected meshed networks*. Taking idea from networking topologies, partial meshed networks are trust networks where each node is at least connected to a subset of nodes it has data exchange with. On the other hand, a fully connected meshed network is a trust network where each node is connected to everyone.

[4] Welkin, http://simile.mit.edu/welkin/

In the former, inferring trust values between a pair of nodes on the network seems difficult but, finding a path between a set of nodes on the network is guaranteed. Using our ontology, recommendations can find efficient paths on the network.

Figure 5 depicts a partial meshed network of people from different contexts and with different goals perhaps, and can be thought of two hybrid networks integrated and merged together. Figure 6 is a visualization of the RDF network for trust network depicted in Figure 5.

VI. STRUCTURAL COMPARISON

In order to emphasize the importance structural determination of trust networks, in this section we consider comparing the structure of the trust networks generated based on three different ontologies; our ontology, Golbeck's and Konfidi's. In the last subsection we discuss in details the results of comparison.

For the sake of comparison, we have divided the experiment datasets into two sizes; small sized networks and large sized networks.

A. Trust networks of small size

Based on our structural point of view, Table 2 lists the number of nodes and edges on the compared networks.

TABLE II
COMPARISON BETWEEN THE SIZES OF SMALL NETWORKS

Trust Networks	Golbeck	Ours	Konfidi
Nodes	15	20	22
Edges	28	34	37

c) Networks of 4 people and 4 relationships. (Increase in size)

Trust Networks	Golbeck	Ours	Konfidi
Nodes	19	28	29
Edges	46	54	58

d) Networks of 4 people and 6 relationships. (Increase in depth)

As it is clear, in general the nodes and edges on the networks generated using Golbeck's ontology is quite smaller than networks generated using our ontology and Konfidi's.

At the same time in both cases our network has a smaller number of nodes and edges than Konfidi's networks, although the difference is not that much.

B. Trust networks of large size

We described and defined hybrid and meshed networks. At the same time, we modeled these networks using datasets that to some extent reflect the structure of such networks. The same datasets were also injected into the structure of two other tested ontologies to consider the structure of the resulting trust networks.

Based on our structural point of view, Table 3 lists the number of nodes and edges on the networks.

TABLE III
COMPARISON BETWEEN THE SIZES OF LARGE NETWORKS

Trust Networks	Golbeck	Ours	Konfidi
Nodes	27	48	50
Edges	73	92	105

a) Hybrid Network (network of 8 people and 12 relationships).

Trust Networks	Golbeck	Ours	Konfidi
Nodes	49	98	86
Edges	132	198	211

b) Meshed network (Networks of 16 people and 26 relationships).

Table 3a shows the number of nodes and edges on the networks representing the hybrid network.

Network generated using Golbeck's ontology has less nodes and edges than both of ours and Konfidi's. Although, network generated using our ontology has less number of edges and nodes in comparison to Konfidi's.

Table 3b shows the number of nodes and edges on the networks representing meshed networks.

Again, Golbeck's network has less number of nodes and edges than our network and Konfidi's network. Our network has greater number of nodes than both, Golbeck's and Konfidi's networks, but lesser number of edges than Konfidi's.

C. Trust networks of larger size

We continued our study by modeling and presenting the trust networks of larger sizes.

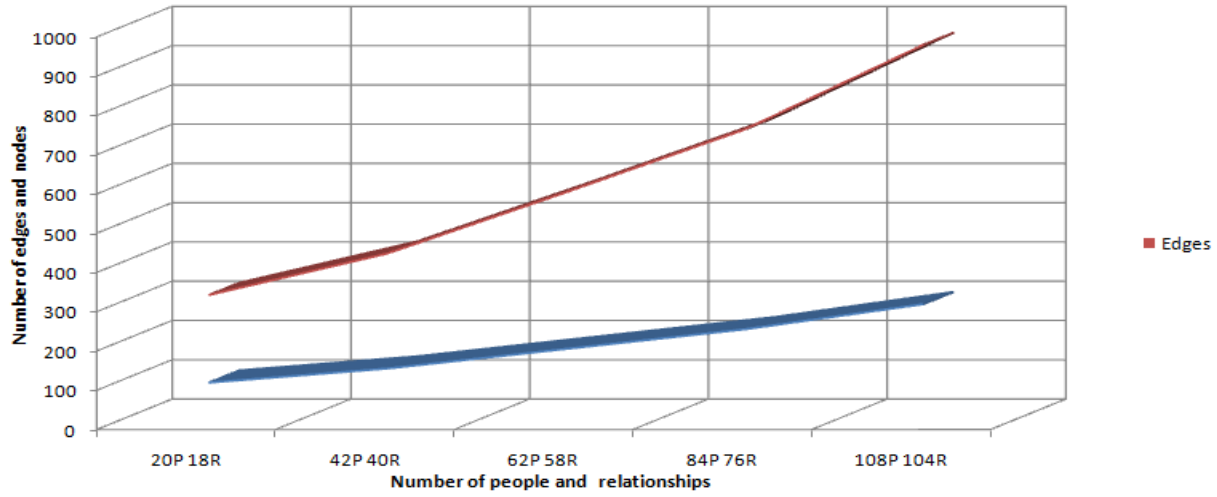
We also expanded our sample partial meshed network and increased the number of people in the networks and their corresponding relationships randomly.

The structure of the resulting networks was studied from the perspective of number of edges and nodes, the same structural perspective used for comparison between networks of small and large size.

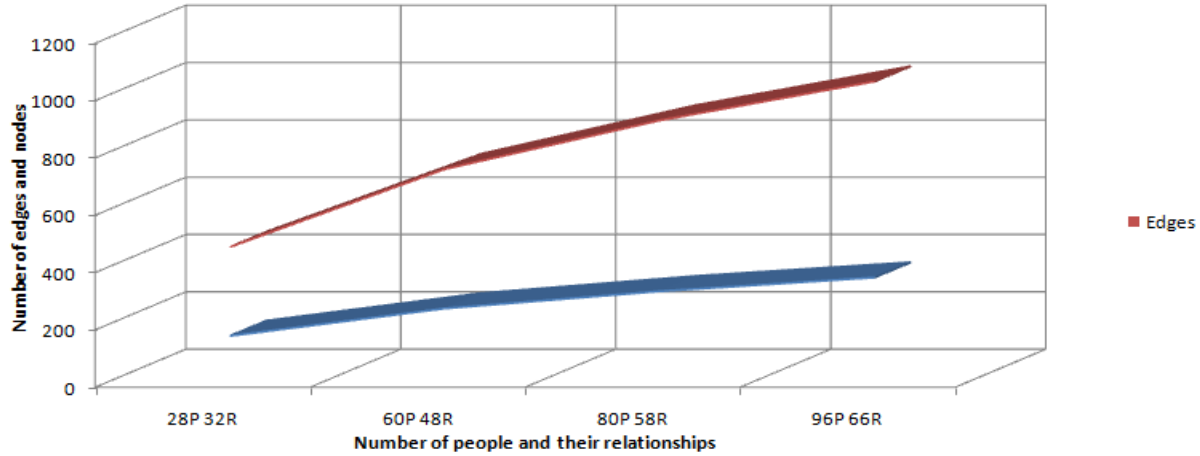
In our experiment we expanded the sample partial meshed network of 16 people and 26 relationships. The number of people and their corresponding relationships were sampled and plotted at each sample increase to reflect the progress of expansion across the network structure.

These data were generated using all three ontologies being evaluated.

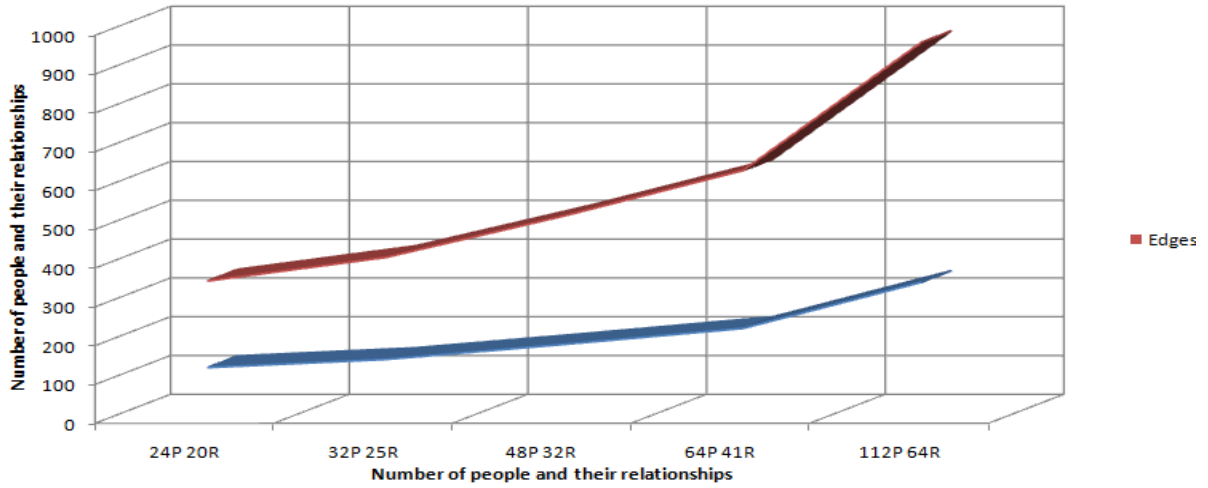
Figure 7, depicts the effect of seamless increase in the size of trust networks of larger size from structural point of view.



a) Increasing the size of Golbeck's trust networks. The diagram depicts the increase in range of nodes and edges, starting from network of 20 people and 18 relations, ending at a network of 108 people and 104 relations.



b) Increasing the size of Konfidi's trust networks. The diagram depicts the increase in range of nodes and edges, starting from network of 28 people and 32 relations, ending at a network of 96 people and 66 relations.



c) Increasing the size of our trust networks. The diagram depicts the increase in range of nodes and edges, starting from network of 24 people and 20 relations, ending at a network of 112 people and 64 relations.

Fig. 7. Networks of larger sizes: Effect of increasing the number of people on the networks described using different ontological structures.

D. Detailed analysis of structural comparisons

In this section we further analyze and study the results of our experiment and comparisons.

As shown in Tables 1 and 2, trust networks modeled, described and presented using our ontology and others are compared based on the number of nodes and edges (structural perspective). Comparison shows that in networks of small size, our ontology shows average performance in comparison to other ontologies, meaning that trust networks generated have average sizes, in comparison. But as the size of the networks increases, certain aspect of trust network size increases more than other compared network, showing less efficient performance. This decrease in efficient performance is also well-depicted in networks of larger size in Figure 7.

There are a set of reasons, which can be stated here.
Clearly, the main reason, for size increase in networks, is the number of elements incorporated within the structure of ontology. Golbeck's ontology uses only one main element, Konfidi uses two main elements, while our ontology uses three main concepts.

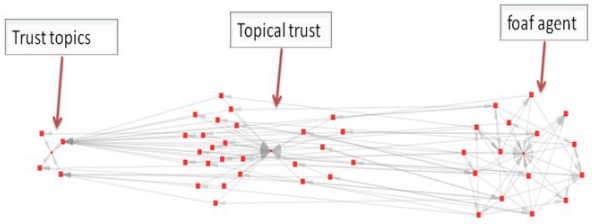


Fig. 7. A clustered visualization of the structure of a meshed trust network based on Jennifer Golbeck's ontology. This network contains 49 nodes and 132 edges.

The second reason would be efficient design of the ontology. Golbeck's ontology is indeed, a mile stone in the work on trust in semantic web, from different perspectives.

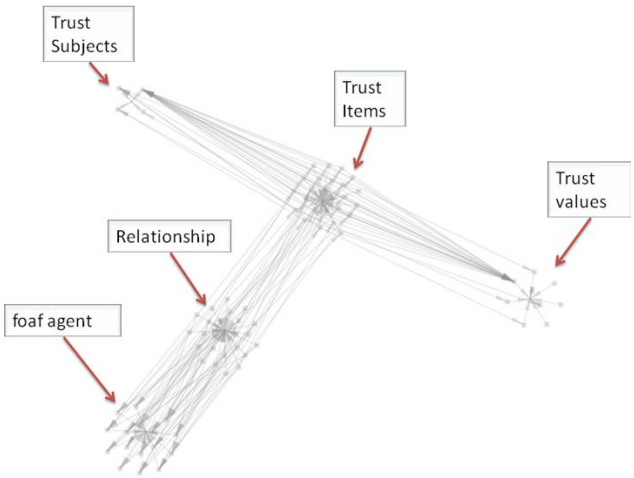


Fig. 8. A clustered visualization of the structure of a meshed trust network based on Konfidi's trust ontology. This network contains 86 nodes and 211 edges.

Her trust schema has a very efficient design. Such design has certain aspects that reduce the size of the networks described using that ontology; first, defining levels of trust (trust0...trust10) and trustRegarding on the range of *foaf:agent* lets you describe the trust directly as the properties of agents and on the trust network. Such efficiency in design lets you describe relations very easily with lesser elements, as seen in results. Konfidi's trust ontology has more or less the same structure like our ontology. Our ontology has one more element than Konfidi's, however we have seen networks of smaller size generated by using our ontology have less complex structures than the ones generated by using Konfidi's ontology.

Figures 9 visualizes the structure of the networks generated using our ontology. The emphasis on the visualizing was put on the gravity of the instances on the network toward their originated main elements. An efficient structure will depict the overall organization of the ecosystem and its sub-ecosystems. Our network shows better clustering of elements among the two other samples.

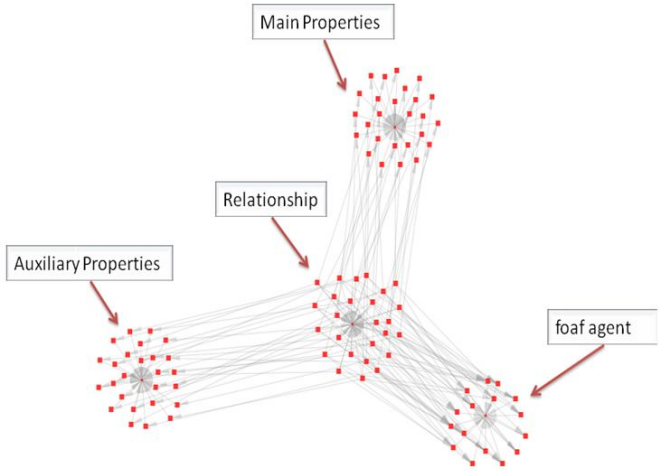


Fig. 9. A clustered visualization of the structure of a meshed trust network based on our trust ontology. This network contains 98 nodes and 198 edges.

The third reason is the *AuxiliaryProperties* element of our ontology. As we incorporated an extensibility element for describing secondary and optional properties, we will incorporate extra nodes and more importantly extra edges into the network. In most of the test data for the comparison section, we have auxiliary property elements with at least one sub-element filled. For instance, when describing hybrid networks, all relationships have *AuxiliaryProperties* with *ContextType* property of either simple social network, or simple business network, or hybrid network. It should be mentioned here that none of the other compared ontologies, have any element for describing extra properties; extending Golbeck's trust ontology seems to be very hard and needs drastic changes because of its architecture, and Konfidi doesn't have any elements for describing extra properties. Taking into account this information, if we eliminate the *AuxiliaryProperties* element, then the size of our network becomes even more efficient than both other ontologies, in certain situations.

VII. CONCLUSION

We analyzed the modeling and representation of trust relationships across the networks within semantic web-driven ecosystems. In order to capture, model and represent the semantics of trust relationships within semantic web, main components of relationships are represented and described using ontologies. To analyze the methodologies and mechanisms used to described trust relations, we studied and analyzed a set of trust ontologies, specially Jennifer Golbeck's and Konfidi's trust ontologies, which share the same context with our research context. At the end, we engineered and analyzed a trust ontology based on the context of our research, social networks and semantic web.

We constructed a trust ontology in which relationship is the focus of ontology, as ontology captures the semantic of trust relationships, and two other elements state the properties of trust relationships. In comparison to previous works, there are certain new features that our work introduces to trust ontologies in this context; using our *AuxiliaryProperties*, we give relationships more weight and meaning. We have introduced the *hasRecommender* property that can determine the strength of the links on social network and can be used for finding the suitable inference path on the network.

We claimed that determining the structure of trust networks could be possible by efficiently designing and engineering trust ontologies that such networks are based upon. We also demonstrated this fact by using the same datasets on both our ontology and two other ontologies. Results of our experiment fairly prove our claim. Having more elements than other ontologies, networks generated based on our ontology show average size and structure. Also our trust networks shows far more manageable structure and architecture as the size increases, in comparison with two other compared ontologies.

As a conclusion, we can state that ontologies are very promising technologies. Utilizing ontologies in modeling and representing trust in semantic web-enabled social systems seems to be a highly efficient methodology and mechanism.

VIII. FUTURE WORK

Studying the social phenomena within computer science and especially semantic web, demands more attention. I believe by having a liaison between social sciences and computer sciences, more fruitful results can be achieved, that can help bringing social ecosystems into life on the web.

Number of vocabularies, used to describe the elements of ontologies should increase. There is a vocabulary to express relationships [48], but there is no standard vocabulary to express for instance, common subjects and topics of a relationship, while we can describe vocabularies using we can easily describe a vocabulary for this matter.

The application domain is very limited and one of the most important future works on this field is spotting certain fields that demands further attention. Current applications are just limited to Spam filtering and user rating systems across web sites on internet.

One of the most important future works is spotting further applications for social trust, where trust relationships can be modeled and expressed using ontologies.

REFERENCES

[1] H. Mariotti, "Autopoiesis, Culture, and Society", (Accessed June 26, 2005). Available at: http://www.oikos.org/mariotti.htm

[2] N. Dokoohaki, M. Matskin, "Structural Determination of Ontology-Driven Trust Networks in Semantic Social Institutions and Ecosystems", Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'07) and the International Conference on Advances in Semantic Processing (SEMAPRO 2007), November 4-9, 2007 - Papeete, French Polynesia. 2007, ISBN 0-7695-2993-3, IEEE Computer Society, Los Alamitos, CA, USA, pp. 263-268.

[3] S. Milgram, "The Small World Problem", Journal of Psychology Today, Vol. 1 (1), pp. 60-67, (1967).

[4] S. Downes, "The Semantic Social Network", published February 14, 2004, (2004). Retrieved from: http://www.downes.ca/post/46

[5] S. Downes, "Semantic Networks and Social Networks". The Learning Organization Journal, Emerald Group Publishing Limited, ISSN 0969-6474, Vol. 12, No. 5, pp. 411-417, (2005).

[6] D. Brickley, L. Miller, "FOAF Vocabulary Specification, Namespace Document", September 2, 2004, Available at: http://xmlns.com/foaf/0.1/

[7] A. L. Cervini, "Network Connections: An Analysis of Social Software That Turns Online Introductions into Offline Interactions", Master's thesis, Interactive Telecommunications Program, New York University. (2003).

[8] Ch. Li, "Profiles: The Real Value of Social Networks", Forrester. July 15, (2004). Available at: http://www.forrester.com/Research/Document/Excerpt/0,7211,34432,00.html

[9] M. Gladwell, "The Tipping Point: How Little Things Can Make a Big Difference", Little, Brown & Company, Boston, MA. ISBN 0-349-11346-7. (2000).

[10] D. Brondsema, A. Schamp, "Konfidi: Trust Networks Using PGP and RDF". Proceedings of the WWW'06 Workshop on Models of Trust for the Web (MTW'06), Edinburgh, Scotland, UK, May 22, 2006 (2006).

[11] J. G. Breslin, A. Harth, U. Bojars, S. Decker, "Towards Semantically-Interlinked Online Communities", Proceedings of the 2nd European Semantic Web Conference (ESWC'05), Springer, (2005).

[12] SIOC (Semantically-Interlinked Online Communities). Available at: http://rdfs.org/sioc/

[13] D. Brickley, S. Stefan, A. Miles, L. Miller, D. O. Caoimh, C. M. Neville, "SIOC Ontology Specification", (2007). Available at: http://rdfs.org/sioc/spec

[14] U. Bojars, J. G. Breslin, A. Passant, "SIOC Ontology: Applications and Implementation Status". W3C Member Submission, (2007). Available at: http://www.w3.org/Submission/sioc-applications/

[15] S. Wasserman, K. Faust, D. Iacobucci, M. Granovetter, "Social Network Analysis: Methods and Applications". Cambridge University Press, ISBN 0-521-38707-8 (1994).

[16] J. P. Scott, "Social Network Analysis: A Handbook". Sage Publications, London. ISBN 0-7619-6338-3 (2000).

[17] I. Cantador, P. Castells, "Multilayered Semantic Social Network Modeling by Ontology-Based User Profiles Clustering: Application to Collaborative Filtering", No. 4248, pp. 334-349, Lecture Notes in Computer Science – Springer, ISSN 0302-9743 (2006).

[18] B. Aleman-Meza, M. Nagarajan, C. Ramakrishnan, L. Ding, P. Kolari, A. P. Sheth, I. Budak Arpinar, A. Joshi, T. Finin, "Semantic Analytics on Social Networks: Experiences in Addressing the Problem of Conflict of Interest Detection". Proceedings of the 15th international conference on World Wide Web, (2006).

[19] P. Mika, “Flink: Semantic Web Technology for the Extraction and Analysis of Social Networks”, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 3, No. 2-3, pp. 211-223, (October 2005).
[20] T.W.A. Grandison, “Trust Management for Internet Applications”. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, Department of Computing, (2001).
[21] T.W.A. Grandison, M. Sloman, “A Survey of Trust in Internet Applications”, IEEE Communications Surveys and Tutorials, 1553-877X, Vol.3, No. 4, Page 2, (2000).
[22] W.V.O. Quine, J. S. Ullian, ”The Web of Belief”, Random House, New York, ISBN 0-394-32179-0, (1978).
[23] L. Ding, T. Finin, “Weaving the Web of Belief into the Semantic Web”, Proceedings of the 13th International World Wide Web Conference, (2004).
[24] A. Abdul-Rahman, S. Hailes, “A Distributed Trust Model”. Proceedings of Workshop on New Security Paradigms. (1998).
[25] J. Huang, M.S. Fox, “An Ontology of Trust – Formal Semantics and Transitivity”, Proceedings of the 8th ACM International Conference on Electronic Commerce. Fredericton, New Brunswick, Canada, pp.: 259 – 270, ISBN 1-59593-392-1, (2006).
[26] PML 2 Trust Ontology. Available at: http://iw.stanford.edu/2006/06/pml-trust.owl
[27] J. Golbeck, B. Parsia, J. Hendler, ”Trust Networks on the Semantic Web”, ISSU 2782, pp. 238-249, Lecture Notes in Computer Science – Springer, (2003).
[28] J. Golbeck, J. Hendler, “Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-based Social Network”, ISSU 3257, pp. 116-131, Lecture Notes in Computer Science – Springer (2004).
[29] J. Golbeck, “Computing and Applying Trust in Web-based Social Networks”, University of Maryland, (2005). Available at: https://drum.umd.edu/dspace/bitstream/1903/2384/1/umi-umd-2244.pdf
[30] J. Golbeck, “Inferring Trust Relationships in Web-based Social Networks”, ACM Transactions on Internet Technology, Vol. 7, No. 1, (2006).
[31] J. Golbeck, “FilmTrust: Movie Recommendations from Semantic Web-based Social Networks”, IEEE Consumer Communications and Networking Conference, (2006).
[32] D.L. McGuiness, P.P. Da Silva, L. Ding, “Proof Markup Language (PML) Primer”, (2005). Available at: http://iw.stanford.edu/2005/wd-pml-primer/
[33] Konfidi, Available at: http://konfidi.org/
[34] G. F. Davis, M. Yoo, W.E. Baker, "The Small World of the American Corporate Elite", Journal of Strategic Organization, Vol. 1, No. 3, pp.301-326, Springer, August 2003, (2003).
[35] C. C. Foster, A. Rapoport, C. J. Orwant, “A Study of a Large Cociogram: Elimination of Free Parameters”. Behavioral Science, Number 8, pp.56-65. (1963).
[36] M. E. J. Newman, "The Structure of Scientific Collaboration Networks," Proceedings of the National Academy of Sciences; 98: 404 - 409. (2001).
[37] D. Watts, “Small Worlds: The Dynamics of Networks between Order and Randomness”. Princeton, NJ: Princeton University Press, ISBN 0-691-00541-9, (1999).
[38] D. Watts, S. H. Strogatz, "Collective Dynamics of Small-World Networks", Journal of Nature, MacMillan Magazines, London, ISSUE 6684, pp. 440-442, (1998).
[39] M. S. Fox, J. Huang, “Knowledge Provenance: An Approach to Modeling and Maintaining the Evolution and Validity of Knowledge”. 22 May 2003, (2003). Retrieved from: http://www.eil.toronto.edu/km/papers/fox-kp1.pdf
[40] J. Huang, M. S. Fox, "Trust Judgment in Knowledge Provenance," DEXA, pp.524-528, 16th International Workshop on Database and Expert Systems Applications (DEXA'05), (2005).
[41] S. Toivonen, G. Denker, ”The Impact of Context on the Trustworthiness of Communication: An Ontological Approach.”, Workshop on Trust, Security, and Reputation on the Semantic Web, (2004).
[42] J. Hradesky, B. Acrement, ”Elements for Building Trust”. Proceedings of iTrust; A Conference on Trust Management. (1994).
[43] Inference Web, Knowledge Systems AI laboratory, Stanford University. Available at: http://iw.stanford.edu/
[44] D.L. McGuiness, P. P. Da Silva, L. Ding, “Proof Markup Language (PML) Primer”, (2007). Available at: http://inference-web.org/2007/primer/
[45] Web of Trust Vocabulary, Version 0.1. Available at: http://xmlns.com/wot/0.1/
[46] D. Brickley, “WOT RDF Vocabulary”. (2002)
[47] I. Zaihrayeu, P. P. Da Silva, D. L. McGuinness, “IWTrust: Improving User Trust in Answers from the Web”. Proceedings of the 3rd International Conference on Trust Management (iTrust). Lecture Notes in Computer Science - Springer. (2005).
[48] I. Davis, Jr. E. Vitiello, “Relationship: A Vocabulary for Describing Relationships Between People”, RDF Vocabulary Specification. (2005). Retrieved from: http://vocab.org/relationship/rel-vocab-20040308.html
[49] J. Davies, R. Studer, P. Warren, “Semantic Web Technologies: Trends and Research in Ontology-based Systems”. John Wiley & Sons, ISBN 0-470-02596-4. (2006).
[50] J. Brank, M. Grobelnik, D. Mladenic, ”A Survey of Ontology Evaluation Techniques”. Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005), Ljubljana, Slovenia. (2005).
[51] A. Jøsang, S. J. Knapskog, “A Metric for Trusted Systems.” Proceedings of the 21st National Security Conference, NSA, October (1998).
[52] J. Avnet, J. Saia, “Towards Robust and Scalable Trust Metrics”. IEEE International Conference, (2003).
[53] G. Caronni, “Walking the Web of Trust”. Proceedings of IEEE 9th International Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises, (WET ICE), ENABL-00, page 153, ISBN 0-7695-0798-0. (2000)
[54] T. Beth, M. Borcherding, B. Klein, “Valuation of Trust in Open Networks”. In Proceedings of the Third European Symposium on Research in Computer Security, ISSU 875, pp. 3–18. Lecture Notes in Computer Science – Springer, (1994)
[55] PR. Zimmermann, “The Official PGP User's Guide”. MIT Press, ISBN 0-262-74017-6, Cambridge, MA, USA. (1995)
[56] Orkut, http://www.orkut.com/.
[57] LinkedIn, http://www.linkedin.com/.

On Choosing a Load-Balancing Algorithm for Parallel Systems with Temporal Constraints

Luís Fernando Orleans[1], Geraldo Zimbrão[1], Pedro Furtado[2]
[1]*COPPE, Department of Computer and Systems Engineering – Federal University of Rio of Janeiro, Brazil*
[2]*CISUC, Department of Informatics Engineering, University of Coimbra, Portugal*
{lforleans, zimbrao}@cos.ufrj.br, pnf@dei.uc.pt

Abstract

A key point in parallel systems design is the way clients requests are forwarded and distributed among the servers, trying to obtain the maximum throughput from them or, in other words, the load-balancing policy. Although it is a largely studied theme, with well accepted solutions, the inclusion of temporal constraints, also denoted as deadlines in this work, to the requests brings new complexities to the load-balancing problem: how to distribute the tasks and minimize the miss rate. The experiments describe along this paper attests that the workload variability plays a crucial role in this problem, pointing the big requests as the most critical elements. Our results also shows that even dynamic load-balancing algorithms are not able to reach an acceptable miss rate, since they handle both short tasks and big tasks the same way. Hence, we propose a new load-balancing algorithm, called ORBITA, which has a request identification and classification mechanism and an admission control module as well, restricting the number of big tasks within the system. This algorithm outperforms its competitors, which means that it has a bigger rate of tasks that end within the deadline, specially when the system is under high load. A prototype was also built in order to check the correctness of the simulation phase. The experiments were run against a benchmark tool, TPC-C, and all the results confirmed the previous assumptions, leading to the conclusion that it is a good practice to understand the system's workload in order to minimize the miss rate.

Keywords: load-balancing, parallel processing, deadline, ORBITA

1. Introduction

In parallel request processing systems, several parallel servers compute the incoming requests (or tasks) that are dispatched to them according to a load-balancing algorithm. Typically, these servers provide no guarantees about the response times for the request executions, in a so-called a *best-effort* approach. In peak situations, with requests arriving at high rates, this policy can lead to a scenario where a request takes tens of times longer to execute than it would take in a less stressed server. This way, if the system provides some kind of quality of service, such as trying to guarantee that response times would not be higher than an acceptable threshold, denoted as deadlines in this paper, the best-effort policy cannot be applied. Guaranteeing acceptable response times in parallel processing systems through load-balancing is the main objective of this paper. Our approach is meant to be applicable in different environments, including Transaction Processing Systems, Web Services, Virtualization Platforms.

This work proposes a new load-balance algorithm, based on the tasks durations (which are supposed to be known *a priori*), and our experiments prove that this is a better approach than blindly dispatching the tasks taking no further considerations – as most load-balance algorithms do. Although there already exists size-aware load-balancing algorithms, such as SITA-E [20], they do not comprise response times concerns, which is responsible for their poor performance on stressed systems with deadlined-tasks.

In order to find the best alternative, we have to analyze the impact of deadlined-tasks and their variability in the known load-balancing techniques. Our approach is valid and performs better than traditional load-balancing ones for both hard or soft deadlines.

The simulated architecture comprises only two servers, because it is the simplest possible parallel architecture. This can be easily expanded to *n* servers as well and this generalization will be discussed throughout the paper.

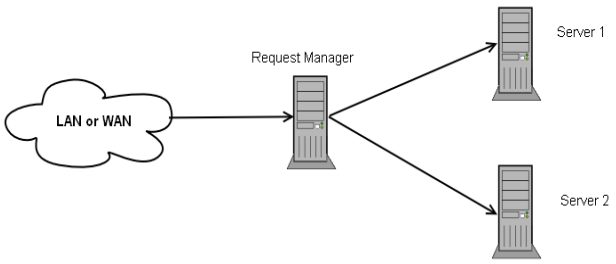


Figure 1: Simulated architecture

2. Related Work

There are plenty of works about load-balancing and QoS, most of them leading to already accepted and consolidated conclusions. Although these are almost exhausted themes, their combination seems to be an area where there are very few research results.

2.1. Load-balancing

Many load-balancing algorithms have been studied, most of them trying to maximize throughput or minimize the mean response time of requests. Reference [19] proposes an algorithm called TAGS, which is supposed to be the best choice when task sizes are unknown and they all follow a heavy-tailed distribution. This is not the case for the scenario analyzed in this paper, in which task sizes must be below a deadline threshold. It is also shown in [19] that, when task sizes are not heavy-tailed, Least Work Remaining has a higher throughput then TAGS. In fact, [24] and [23] claim that Least-Work Remaining is optimal when task sizes are exponential and unknown.

The algorithm SITA-E [20] has the best performance when task sizes are known and heavy-tailed but, otherwise, Least-Work-Remaining presents a better throughput.

Our previous work in [25] presented a technique to determine the best multiprogramming level (MPL) *offline*. Such concept had been expanded and we propose an algorithm that computes the maximum MPL in *runtime*.

2.2. Quality-of-Service (QoS)

In real distributed systems, task sizes are heavy-tailed. This means that a very small portion of all tasks are responsible for half of the load [21]. Most tasks are very small and there is a small number of big tasks as well. In models with deadlines, like the one analyzed in this paper, a similar distribution occurs.

Reference [30] presents a model where the number of concurrent requests within the system is restricted. When this number is reached, the subsequent requests are enqueued. But this model has no concern for deadlines or rejection of requests. It also does not show a way to load-balance the arriving tasks, since it is a single-server architecture.

Quality-of-Service was also studied for Web Servers. In [9] the authors propose session-based Admission Control (SBAC), noting that longer sessions may result in purchases and therefore should not be discriminated in overloaded conditions. They propose self-tunable admission control based on hybrid or predictive strategies. Reference [8] uses a rather complex analytical model to perform admission control. There are also approaches proposing some kind of service differentiation: [5] proposes architecture for Web servers with differentiated services; [6] defines two priority classes for requests, with corresponding queues and admission control over those queues. In [30], the authors propose an approach for Web Servers to adapt automatically to changing workload characteristics and [14] proposes a strategy that improves the service to requests using statistical characterization of those requests and services.

Comparing to our own work, the load-balancing alternatives referred above do not consider QoS parameters, such as deadlines, and the QoS studies concern single server systems, only. Our approach uses multiple servers and a careful request allocation to those servers in order to comply with the deadline constraints.

3. Modelling typical real distributions

In order to analyze and propose time-considering load-balance approaches, it is important to understand first the kinds of workloads distributions that happen typically and how to model them. The typical request workload, such as Transaction Processing Systems, is quite heterogeneous in what concerns servicing requirements.

Besides the algorithm SITA-E, reference [20] presents a study that claims that the distribution of task sizes (or durations) in computer applications are not exponential, but heavy-tailed. In short, a heavy-tailed distribution follows three properties:

1. Decreasing failure rate: the longer a task runs, the longer it is expected to continue running.
2. Infinite variance
3. A very small fraction (< 1%) of the very largest tasks makes up a large fraction (50%) of the load. This property is often called as the *heavy-tailed property*.

The simplest heavy-tailed distribution is the *Pareto* distribution, with probability mass function:

$$f(x) = ak^a x^{-a-1}, a, k > 0, x > k \quad ,$$

and cumulative distribution function

$$F(x) = 1 - (k \div x)^a \quad .$$

In these functions, k is the smallest possible observation, whereas a is the exponent of the power law, and will be called hereafter as the *variance factor* of the function. It varies from 0 to 2 and the more it is close to 0, the greater is the variability.

4. Traditional load-balancing algorithms and their weakness

Load balancing is a fundamental basic building block

for construction of scalable systems with multiple processing elements. There are several proposed load-balancing algorithms, but one of the most common in practice is also one of the simplest ones - Round-Robin (RR). This algorithm produces a *static* load-balancing functionality, as the tasks are distributed round-the-table with no further considerations. At the other hand, the algorithm Least-Work-Remaining (LWR) produces a *dynamic* load-balancing, as the arriving tasks are dispatched to the server with the least utilization (jobs on queue or concurrent executing tasks). This algorithm is considered in this study, as it is supposed to be the best choice when tasks durations are not heavy-tailed. According to [19] and [20], the main issue involving these algorithms is the absence of a control over big tasks, mixing inside the same server short (small) and long (big) tasks. It becomes more evident when tasks durations are heavy-tailed, with a minuscule fraction of the incoming tasks being responsible for half of the load. In fact, both [19] and [20] propose size-aware algorithms, trying to minimize the effects caused by the big tasks on the small ones.

We are concerned with guaranteeing specified acceptable response time limits. The number of concurrent executions (CE) is a crucial variable when deadlines are involved, because as we increase the number of CE we have a larger probability of missing the deadlines. As we are going to see, this is an issue that affects mostly systems where tasks durations have a high variability, which means that the occurrence of big tasks is more usual. When the variability is low, i.e., the number of big tasks is near to zero, all algorithms have similar performance curves and practically all tasks are completed. As performance starts to degrade as the variability begins to increase, the number of canceled tasks also gets higher, which gives space for a new size-aware load-balance algorithm, On-demand Restriction for Big Tasks, or ORBITA in short. The main idea is to separate the short tasks, which will always be submitted to execution, from the big tasks, which will have their admission by a server (or node) dynamically controlled. This way, a node will only admit big tasks that will not make the other already running big tasks miss their deadlines. Otherwise, the big task will be rejected by the node, as its admittance would lead to further performance degradation. In an *n* servers scenario, a task is only rejected if none of the *n* servers is able to handle it.

In the following we describe each load balancing algorithm we compare considering deadlines and rejection:

4.1. Least-Work-Remaining (LWR)

```
for each task that arrives:
  next_server := server_list ->least_utilized
  send (task, next_server)
```

4.2. Task Assignment by Guessing Size (TAGS)

In this algorithm, all incoming tasks are dispatched to the first server. If a task is running for too long, i.e., is a big task, it is killed and restarted from scratch on the second server.

```
for each task that arrives:
  send (task, first_server)
  schedule_dispatch_to_second_server(task)
```

4.3. Size Interval Task Assignment with Equal Load (SITA-E)

```
for each task that arrives:
  if task is a big task
    server := server_list ->second_server
  else
    server := server_list ->first_server
  send (task, server)
```

4.4. On-demand Restrictions for Big Tasks (ORBITA)

```
for each task that arrives:
  if task is a small task
    server := server_list ->first_server
    send(task, server)
  else
    server := server_list ->second_server
    bigger_task := bigger_running_task(server)
    max_ce := LOWER_BOUND(deadline/bigger_task)
    if number_of_running_tasks(server) >= max_ce
      NOT_ADMITT(task)
    else
      send (task, server)
```

Figure 2 depicts how LWR and SITA-E behave. In the figure, a new job with estimated duration of 3 units of time (UT) is received (2a) and there are 2servers: one is executing 3 short tasks and the other one is executing 2 long tasks. If the load-balancer module, the light-gray rectangle in figures, is using a LWR strategy, then the new task is dispatched to the second server (2b). On the other hand, if a size-aware algorithm (like SITA-E) is used, the job is forwarded to the first server, the one containing short tasks (2c).

5. Simulation setup

Due to the large number of parameters involved, it becomes necessary to formally describe the simulation model used in this work. The simulator has the following parameters :

- Number of servers.
- Tasks arrival rate (follows an exponential distribution).

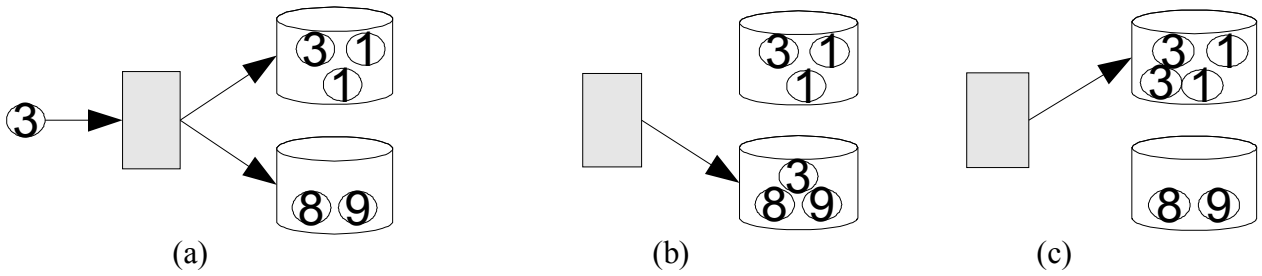


Figure 2: A new transaction arrives (a). LWR approach (b). Size-aware approach(c).

Table 1: Percentage of generated durations, according to the variance factor (*a*) of the Pareto distribution and the duration inteval.

a	[0, 1[	[1, 2[	[2, 3[	[3, 4[	[4, 5[	[5, 6[	[6, 7[	[7, 8[	[8, 9[	[9, 10[	10
0,1	53,88%	6,97%	5,17%	4,90%	4,42%	4,64%	3,93%	4,24%	4,25%	3,74%	3,85%
0,2	76,02%	4,75%	3,18%	2,94%	2,30%	1,90%	1,87%	1,95%	1,79%	1,56%	1,75%
0,3	87,90%	2,91%	1,76%	1,38%	1,12%	0,98%	0,89%	0,80%	0,78%	0,75%	0,73%

- Task size distribution (follows a Pareto distribution).
- Maximum amount of time a task can execute (deadline).
- Load-balancing algorithm.
- Minimum size of "big tasks".

In this paper, a system with 2 identical servers was simulated. The deadline time was set to 20 seconds and the duration of each request follows a Pareto distribution, where the value of the parameter *a* varies from 0.1 through 0.3, step 0.1, the smallest possible duration is 0.001 second (1 millisecond) and the highest duration is 10 seconds. In addition, all tasks have the same priority. The concurrency model is linear, which means that a task will take twice longer if it shares the server with another task, it will take three times longer in case of two other tasks and so forth.

The simulator implements all the described algorithms: TAGS, SITA-E, LWR and ORBITA and task arrivals follow an exponential distribution, with λ varying from 1 to 10, step 1. Finally, the tasks which have their durations below 1 second are considered small tasks. The big tasks are constituted by all the other durations. To eliminate the transient phase, the data obtained in the first hour of the simulation was discarded. Only the results obtained in the next 5 hours were considered.

5.1. Task duration generation

Since we are simulating a scenario where tasks have deadlines, the variance is not infinite. But as we are interested in studying how the system behaves when the number of small tasks is much greater than the number of big tasks, the Pareto distribution is used to generate the duration of the tasks. The MOD function will be applied to all durations that exceed the deadline time (generated_duration MOD deadline), in order to equally distribute them among the allowed durations. Table 1 shows the percentage of the generated durations for values of *a* versus the intervals (that must be read as [*min*, *max*[). It is to notice that when *a* assumes lower values, the variability is higher. Even in these cases, the number of durations within the interval [0,1[is much greater than the others.

5.2. Simulation Results

In these experimental results, we analyze first results for all tasks, showing that ORBITA has better or at least as good performance as other approaches in that case, and almost zero miss rates unlike the other strategies. The big tasks are a small fraction of the workload, but they are the ones with the largest miss rates for most strategies, and that is where ORBITA obtains much better results than the other ones because it considers time constraints. For this reason we then analyze results concerning the big tasks.

Figure 4a shows the performance of the algorithms when tasks durations highly vary. It can be noticed that, as the arrival rate raises, both LWR and TAGS performs worse in comparison with the other two algorithms, SITA-E and ORBITA. This low performance occurs due to the fact that those algorithms mix small and big tasks inside the same server, while SITA-E and ORBITA reserve a server to execute small, fast requests. A quick look to figure 5a attests this explanation: LWR and TAGS have small tasks getting canceled, an event that does not occur nor in SITA-E neither in ORBITA. Even a high arrival rate such as 10 tasks per second does not make the small tasks miss their deadlines when those algorithms are used.

If the variance factor was set to 0.3, the throughput of the four strategies would be very similar, as shown in figure 4b. As this variance factor generates a smaller number of big tasks, the assumption that mixing all kind

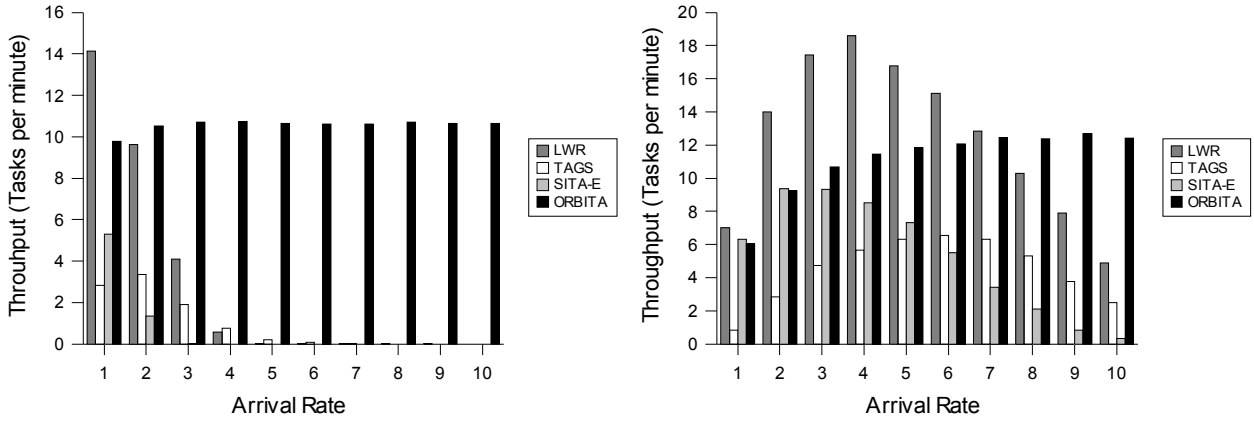


Figure 3: Throughput of big tasks with variance factor 0.1 (a) and 0.3 (b).

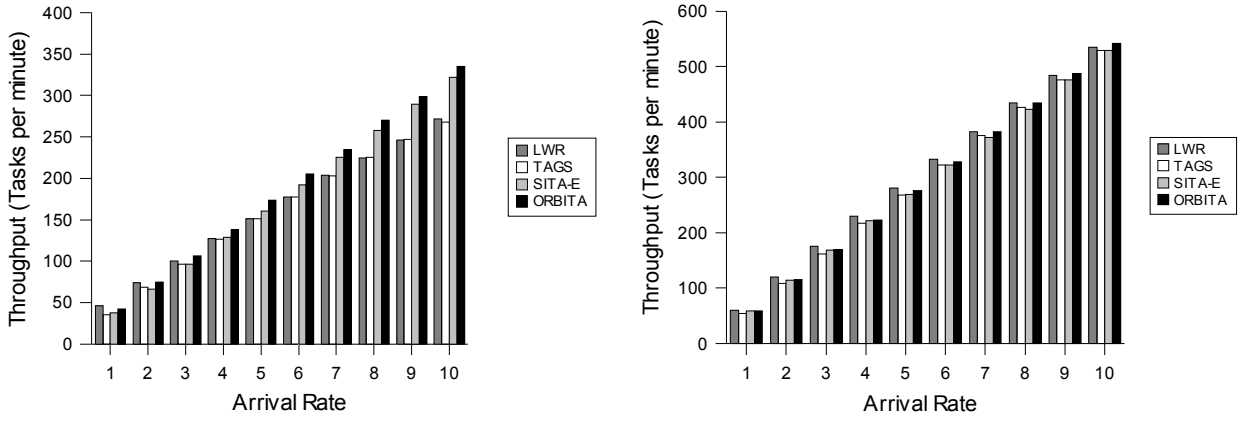


Figure 4: Total throughput with variance factor 0.1 (a) and 0.3 (b).

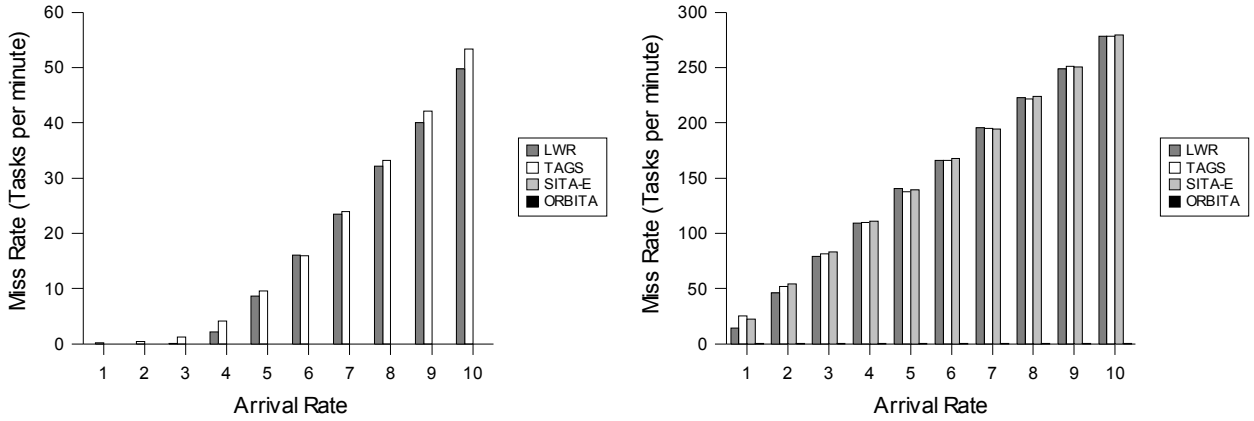


Figure 5: Throughput of small tasks with variance factor 0.1 (a) and 0.3 (b).

of tasks inside the same server for highly heterogeneous workloads is a bad idea, as shown in figure 4a, is reinforced.

It is worth noticing that in both cases in figure 4, the ORBITA algorithm has a better performance than its competitors. It becomes more evident if we analyze the throughput of small tasks and big tasks in separated graphics (figures 3 and 5).

As discussed above, both ORBITA and SITA-E policies prevent small tasks from being killed. This statement can be observed in figure 5a. A consequence of this event is that if we expand the simulated architecture from 2 to *n* parallel servers, only 1 server should be sufficient to handle all small tasks, with the other (*n-1*) servers being used to handle the big tasks. If the arrival rate keeps growing, there will be a moment when the number of servers destinated to handle small tasks should also increase. But the point is that the number of nodes that should be used to handle big tasks is much more critical than those that should be dedicated to small tasks.

The graphics shown in figures 3a and 3b shows the throughput of the big tasks. These pictures confirm the robustness of ORBITA, which maintains the throughput of big tasks almost unaltered, even when the variance factor is 0.1 and the arrival rate is 10 tasks per second.

Figure 3a shows that, for highly heterogeneous workloads, ORBITA is the only strategy with satisfactory results.

In figure 3b, the algorithm LWR presents a better throughput than ORBITA for arrival rates below 7 tasks per second. A closer look at the ended tasks of each strategy, displayed in tables 2 and 3, shows that LWR strategy has a better throughput for task arrival rates comprehended between 1 and 6, and ORBITA presents a better throughput for arrival rates higher than 7 tasks per second. Considering results of figures 3a and 3b together, we conclude that ORBITA is better with highly heterogeneous workloads or high arrival rates due to its tight control on time constraints. LWR is also an interesting algorithm as it tries to optimize resource usage, but does not have enough control over time constraints. Our current and future work on this issue, involves considering adaptable ORBITA/LWR alternatives and LWR with time constraints.

6. Prototype experiments

The Midas middleware [29] is a tool that intercepts the requests sent by an application to a database server. It uses the Proxy Design Pattern, thus providing a transparent admission control layer, without requiring deep source code modifications. A simplified class-diagram is shown in figure 6.

The utilization of more than one database server opens the necessity to keep data synchronized on all nodes, an issue that is known as data replication. According to [35], there are two replication models: eager replication, where the updated data is synchronized at the other nodes before transaction completion, and lazy replication, where the updated data is sent to other servers after transaction completion. For simplicity, we used a primary copy replication strategy, since reference [35] attests it is the best choice for eager replication.

7. Experiment Setup

To better understand the implications of ACID properties on load-balancing algorithms, we performed various rounds of experiments using the TPC-C

Table 2: Throughput histogram of LWR algorithm with variance factor 0.3.

Arrival Rate	[0, 1 [	[1, 2 [	[2, 3 [	[3, 4 [	[4, 5 [	[5, 6 [	[6, 7 [	[7, 8 [	[8, 9 [	[9, 10 [	10
1	52,85	1,77	1,09	0,78	0,65	0,5	0,57	0,45	0,41	0,43	0,36
2	106,56	3,62	2,16	1,66	1,39	1,2	0,98	0,97	0,82	0,63	0,58
3	157,52	5,16	3,08	2,54	1,91	1,34	1,15	0,78	0,69	0,46	0,33
4	210,77	7,31	4,4	2,86	1,58	0,96	0,57	0,36	0,25	0,16	0,13
5	264,27	9,19	4,51	1,74	0,7	0,27	0,17	0,09	0,05	0,02	0,02
6	317,21	10,58	3,47	0,79	0,19	0,06	0,02	0,02	0	0	0
7	369,7	11,01	1,63	0,16	0,02	0,01	0	0	0	0	0
8	423,7	9,71	0,56	0,03	0	0	0	0	0	0	0
9	476,46	7,76	0,16	0	0	0	0	0	0	0	0
10	529,75	4,88	0,02	0	0	0	0	0	0	0	0

Table 3: Throughput histogram of ORBITA algorithm with variance factor 0.3.

Arrival Rate	[0, 1 [	[1, 2 [	[2, 3 [	[3, 4 [	[4, 5 [	[5, 6 [	[6, 7 [	[7, 8 [	[8, 9 [	[9, 10 [	10
1	53,06	1,43	0,91	0,69	0,53	0,5	0,44	0,41	0,41	0,37	0,37
2	106,07	2,31	1,29	1,11	0,87	0,81	0,64	0,65	0,57	0,55	0,45
3	159,01	2,6	1,38	1,23	1,03	0,97	0,82	0,71	0,68	0,68	0,59
4	211,17	2,92	1,73	1,2	1,05	0,95	0,85	0,8	0,68	0,64	0,65
5	263,86	2,9	1,85	1,24	1,02	1,08	0,86	0,72	0,79	0,65	0,73
6	316,49	3,02	1,75	1,3	1,12	0,94	0,94	0,9	0,71	0,67	0,72
7	370,23	3,21	1,98	1,35	1,11	0,97	0,88	0,75	0,76	0,73	0,71
8	422,01	3,08	1,75	1,47	1,07	1,05	0,9	0,87	0,81	0,7	0,68
9	474,35	3,27	1,98	1,28	1,24	1,08	0,9	0,83	0,74	0,7	0,68
10	529,24	3,13	1,8	1,29	1,11	1,02	0,93	0,91	0,82	0,72	0,68

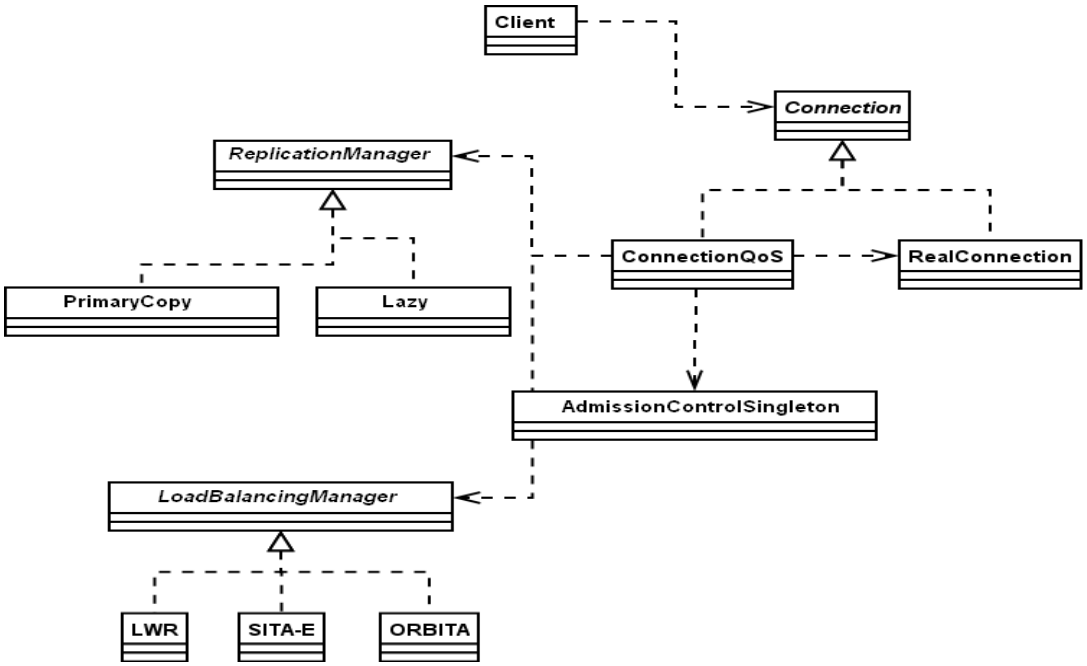


Figure 6: Simplified class-diagram of the Midas middleware.

benchmark (http://www.tpc.org/tpcc/) in a 2-servers full-replicated database.

Also, to create a more realistic scenario, we modified the workload generation. According to related work [19], typical transactional workloads present high-tailed properties, which is not the case of TPC-C's default workload. We also made an extra modification on the TPC-C's code, simulating an open model, i.e., transactions are sent to the system according to a statistical distribution (e.g. an Exponential Distribution). For more informations about open and closed simulation models, please refer to [32].

The use of an open model was a bit imprecise due to the high cost to create and destroy threads (for emulating clients). For arrival rates greater than 14 transactions per second, the mean value of the exponential distribution was not able to be reached. Nevertheless, the obtained results clearly simulated high-utilization scenarios, the aim of the experiments.

7.1. Workload Generation

As mentioned before, we used 2 types of transaction mixes: the one described in the TPC-C specification (which we call default transaction mix in this paper) and another one that aims to reflect a more realistic one, as stated in [19].(denoted here as heavy-tailed transaction mix).

Briefly explaining, the TPC-C specification proposes 5 different transactions (in parenthesis are their frequency of occurrence): new order (45%), payment (43%), delivery (4%), order status (4%) and stock level (4%). When executing in standalone mode, we found that new order transaction was the longest one and, in contrast, the stock level was the fastest transaction. So, we create the heavy-tailed transaction mix comprising only stock level (95%) and new order (5%) transactions, thus attending the requirements of a heavy-tailed distribution.

To classify transactions in short or long we used a map. As TPC-C has only five different transactions, we could store pounded mean response times (PMRT) for each one of them. Hence, each map entry was a pair {transaction name, PMRT}. If the PMRT value was greater (lower) than a threshold (1 second, arbitrarily chosen) then its corresponding transaction would be classified as long (short).

A last remark on the workloads: order status and stock level transactions are *read-only*, which means that they should not acquire any locks. On the other hand, the others are update transactions and their isolation levels were set to *Read-Committed.*

7.2. Experiments Details

All experiments were executed using a Pentium 4 3.2GHz, with 2GB RAM DDR2 and a 200 GB SATA HD which was responsible for creating the threads that simulate the clients. The servers were 2 Pentium II MMX 350MHz, with 256MB RAM and a 60GB IDE HD. Both servers were running a Debian Linux, with Kernel version 2.6 and were connected by a full-duplex 100Mbps Ethernet link. A PostgreSQL 8.1 database server was running on each server machine and the database size was 1.11GB. The client machine used a

Sun Microsystems' Java Virtual Machine, version 1.5. The database was created with 10 warehouses.

The reason for using the slower computers as the database servers relies on our need to stress the system. Our intention is not to maximize the throughput within deadline (TWD) – the most important metric for this work, but to analyze the impact of ACID properties on load-balancing algorithms and their implications on QoS constraints.

Table 4: Simulation parameters and their values

Parameter	Value
Arrival rate	Exponential distribution
Simulation time	20 minutes
Warm-up time	5 minutes
Deadline	5 seconds

Finally, each round of experiments was executed for a period of 20 minutes. During the first 5 minutes no data was collected. Before each round, the database was dropped and then recreated, guaranteeing that all rounds of experiments used the same database state.

7.3. Results

Figures 7 and 8 show the throughput of transactions that ended within the deadline versus the arrival rate. For best visualization, the results are displayed in terms of long transactions (a) and short transactions (b).

It can be seen that when the default workload is used (figures 7a and 7b), LWR performs badly. As a result for mixing inside the same server both short and long transactions, only those which are the fastest ones are able to end within the deadline. This can be explained by the high number of locks obtained by update transactions (the majority on this workload), interfering on the execution of read-only transactions.

On the other hand, SITA-E and ORBITA performs better due to their size-aware nature. Such algorithms estimate the duration of incoming transactions and, if classified as long (short), the transaction is forwarded to the appropriate server. The thing to be noticed here is that short transactions are the read-only ones, thus they do no acquire locks. This explains the high number of short transactions that were executed within the deadline with these algorithms. On the other hand, the TWD of update transactions is smaller, and for SITA-E it even presents a decreasing form on figure 7a. Why does ORBITA perform better than SITA-E? The reason is the admission control that ORBITA provides for long (big) transactions – which are responsible for acquiring locks and, in these cases, for a long period of time. Controlling the admission of long transactions directly implies in controlling the number of locks, thus keeping the accepted transactions ending their executions within their deadlines. It can be seen from figure 7a that an arrival rate of 10 transactions per second reaches the optimal TWD.

A last analysis of figure 7 that is worth mentioning is about replication. Once all update transactions are replicated to others servers, why the performance of small transactions was not affected by the replication in SITA-E and ORBITA? The most reasonable answer for this question relies on the assumption that only the write-set (WS) of update transactions were forwarded to the other server. These WS's do not contain any 'select' statements, thus their execution is also very fast – which means that their executions do slow down the read-only transaction, but are not capable for making them to miss their deadlines.

Figures 8a and 8b present the performance of the same algorithms when a heavy-tailed workload is used. One of the properties of such a distribution is massive presence of short transactions and only a few of very, very big transactions. The results in these figures are interesting, because the presence of only 5% of new order transactions is sufficient to reduce a lot the TWD of short transactions when the LWR algorithm was chosen. In fact, when the arrival rate is 6 (which means 360 transactions per minute, 95% of them are short), TWD reached its maximum value, about 100. As both servers have to execute new order transactions, locks used by this transaction (which includes a 'select for update' clause) also occur everywhere. Again, the TWD of long transactions is zero – even for small arrival rates. This can be ironically explained by the massive presence of fast transactions, which severely interferes on the executions of long transactions. So, long transactions do not permit that the short ones execute within deadline. In turn, the presence of lots of short transactions are responsible for the zero-TWD of long jobs.

The other algorithms, SITA-E and ORBITA, have better performances. As in the default workload setup, none of the small transactions missed their deadlines, even when data modifications provided by the replications occur. On the other hand, the execution of long transactions for SITA-E becomes critical. As the arrival rate increases, the TWD fastly decreases until it reaches zero. Again, the high number of locks are responsible for the poor performance. In contrast to SITA-E, ORBITA is capable to maintain the TWD of long transactions in its maximum, due to its admission control mechanism. A remark about this admission control relies on the reduced number of long transactions admitted. When TWD reached its maximum value, the arrival rate is about 10, what means that 600 transactions arrives per minute – 5% of them (about 30) are of new order type – but only 7 of them were admitted. Otherwise, there might be conflicts on lock acquisitions and, probably, some transactions (maybe even all of them) would miss their deadlines.

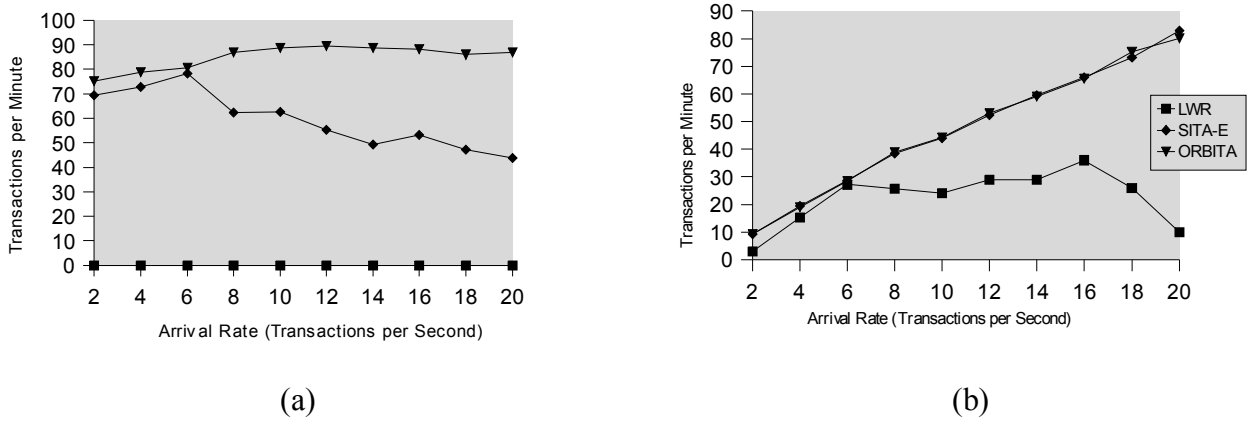


Figure 7: Throughputs of long transactions (a) and small transactions (b) when default workload is used.

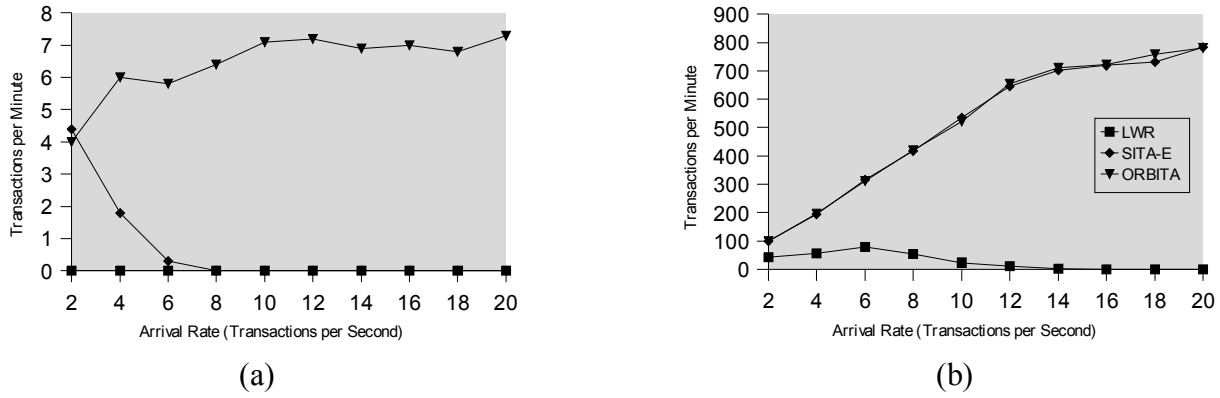


Figure. 8: Throughputs of long transactions (a) and small transactions (b) when heavy-tailed workload is used.

8. Conclusion and future work

In a parallel system, the incoming tasks must be dispatched to a server according to a load-balancing algorithm. Most of these algorithms are not aware about the response times of the tasks, since they all follow a best-effort policy. Thus, if tasks arrive at a very high rate, not only fast requests will have to wait for a long period to be executed, as slower requests will take too long. In this paper we propose ORBITA, a load-balance algorithm that takes time into consideration.

The ORBITA algorithm, which is based on the assumption that tasks durations follow a highly heterogeneous distribution, differentiates service between small and large requests in order to provide time guarantees. It was experimentally proved that the big tasks were the ones responsible for deadline misses, so ORBITA works by separating the fast, small tasks from the big tasks, which have their admission controlled by each server. A big task will only be admitted into a server if it does not make the other running big tasks miss their deadlines.

The experiments have shown that when the variability of the task durations is high, ORBITA's throughput is not only greater than the other algorithms, but fairer, since tasks of all size intervals have low miss rates.

We also implemented a prototype containing three different of the presented load-balancing techniques, each one with different characteristics: Least-Work-Remaining (LWR), Size Interval for Task Assignment with Equal Load (SITA-E) and On-Demand Restrictions for Big Tasks (ORBITA), our proposal.

Those rounds of experiments were executed in a 2-servers fully replicated database. The dynamic algorithm (LWR) presented the worst performance, since it does not make differentiations on transactions. Thus, by mixing inside the same server update and read-only transactions, the isolation needed for the first group was responsible for making lots of transactions (of both groups) to miss their deadlines.

On the other hand, the size-aware algorithms (SITA-E and ORBITA) dynamically recognized read-only (update) transactions and classified them in short (long). This way, each group was assigned to a dedicated server, hence reducing the overall miss rate. Even the replication cost was not sufficient for deteriorate the performance of read-only transactions, since only the write-set of the whole update transaction was forwarded to the other server. The main problem for SITA-E relies on the execution of long transactions, which are responsible for acquiring the locks. The uncontrolled admission of update transactions present in the SITA-E can reduce the number of transactions ended within the deadline per minute to zero. In contrast, ORBITA has an admission control of big transactions. An update transaction is only admitted by the system if it will not cause deadline misses – from itself or from the other already-running transactions.

As future works, we intend to investigate how to effectively identify and estimate the duration of transactions. The solution adopted in this paper (using a map with pounded mean response times) was sufficient for what this work was intended, but we are concerned on if and how to generalize such a concept for a system with ad-hoc transactions. We also intend to work on database internals level and study the viability of adding time-constraints mechanisms to queries and/or transactions.

As future work we intend to investigate how ORBITA can be made to adapt automatically to actual workloads and arrival rates. This includes how to determine the number of servers needed to handle each type of tasks (big and small) and, instead of dividing tasks into two classes statically, how to determine and manage task classes automatically. We also intend to work on a time-constrained version of LWR and compare the approaches.

References

[1] Akal, F. et al (2005), Fine-Grained Replication and Scheduling with Freshness and Correctness Guarantees. 31st Very Large Databases Conference, p. 565 – 576, Trondheim, Norway, 2005.

[2] Amza, C., Cox, A.L., Zwaenepoel, W. (2005), A Comparative Evalution of Transparent Scaling Techniques for Dynamic Content Servers. 21st International Conference On Data Engineering, 2005.

[3] Barker, K. et al (2004), A Load-Balancing Framework for Adaptative and Asynchronous Applications, IEEE Journal Transactions on Parallel And Distributed Systems, v.15, n.2.

[4] Barker, K.; Chernikov, A.; Crhisochoides, N.; Pingali, K. (2004), A Load Balancing Framework for Adaptive and Asynchronous Applications. IEEE Transactions on Parallel and Distributed Systems, v. 15, n. 2

[5] Bhatti, N.; Friedrich, R. (1999) Web server support for tiered services. IEEE Network, v.13, n.5, p. 64–71.

[6] Bhoj; Rmanathan; Singhal. (2000) Web2K: Bringing QoS to Web servers. Tech. Rep. HPL-2000-61, HP Labs.

[7] Cardellini, V.; Colajanni, M.; Yu, P.S. (2002), The State of the Art in Locally Distributed Web-Server Systems. ACM Computing Surveys, v. 34, p.263 – 311

[8] Chen, X; Mohaptra, P; Chen, H.(2001), An admission control scheme for predictable server response time forWeb accesses. In Proceedings of the 10th World Wide Web Conference, Hong Kong.

[9] Cherkasova; Phall (2002), Session-based admission control: A mechanism for peak load management of commercial Web sites. IEEE Req. on Computers, v.51, n.6.

[10] Crovella, M.; Bestravos, A. (1997), A.Self-similarity in World Wide Web traffic: Evidence and possible causes. IEEE/ACM Transactions on Networking, p.835-836.

[11] Daudjee, K.; Salem, K. (2006), Lazy Database Replication with Snapshot Isolation, 30th Very Large Databases Conference, p. 715 – 726, Seoul, Korea.

[12] Devine, D.K. et al (2005), New challenges in dynamic load balancing,ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation, v. 52, n. 2-3, p. 133 – 152

[13] Dyachuk, D.; Deters, R. (2007), Optimizing Performance of Web Service Providers, IEEE 21st International Conference on Advanced Information Networking and Applications, p. 46 – 53, Niagara Falls, Ontario, Canada.

[14] Elnikety, S.; Nahum, E.; Tracey, J; Zwaenepoel, W. (2004) A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites, WWW2004: The Thirteenth International World Wide Web Conference, New York City, NY, USA.

[15] Furtado, P.; Santos, C. (2007), Extensible Contract Broker for Performance Differentiation, International Workshop on Software Engineering for Adaptive and Self-Managing Systems, Minneapolis, USA.

[16] Gamma, E. et al (1994), Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

[17] Ghazalie, T. M.; Baker (1995), T. P. Aperiodic Servers In A Deadline Scheduling Environment. Real Time Systems Journal. v 9, n. 1, p. 31 – 67

[18] Gokhale, S. S.; Lu, J. (2006), Performance and Availability Analysis of E-Commerce Site, 30th Annual International Computer Software and Applications Conference, Chicago.

[19] Harchol-Balter, M. (2002), Task assignment with unknown duration.. Journal of the ACM

[20] Harchol-Balter, M.; Crovella, M.: Murta, C. (1999), On choosing a task assignment policy for a distributed server system. Journal of Parallel and Distributed Computing, v.59 n.2, 204-228.

[21] Harchol-Balter, M.; Downey, A. (1997), Exploiting process lifetime distributions for dynamic load-balancing. ACM Transactions on Computer Systems.

[22] Knightly, E.; Shroff, N. (1999), Admission Control for Statistical QoS: Theory and Practice. IEEE Network, v. 13, n. 2, pp. 20-29.

[23] Nelson, R.; Philips, T. (1993), An approximation for the mean response time for shortest queue routing with general interarrival and service times. Performance Evaluation, p.123-139.

[24] Nelson, R.; Philips, T. (1989), An approximation to the response time for shortest queue routing. Performance Evaluation Review, p.181-189.

[25] Orleans, L.F., Furtado, P.N. (2007), Optimization for QoS on Web-Service-Based Systems with Tasks Deadlines, ICAS'07 Third International Conference on Autonomic and Autonomous Systems, 2007.

[26] Orleans, L.F., Furtado (2007), P.N., Fair Load-Balancing on Parallel Systems for QoS, ICPP International Conference on Parallel Processing, p. 22

[27] Orleans, L.F., Zimbrão, G., Furtado, P.N. (2008), Controlling the Behaviour of Database Servers with 2PAC and DiffServ, DEXA'08 - 19th International Conference, DEXA 2008, Turin, Italy

[28] Orleans, L.F., Zimbrão, G., Oliveira, C.E.T. (2008), On Choosing a Load-Balancing Algorithm for Parallel Databases with Time-Constraints, SBBD'08, XXIII Brazilian Symposium on Databases, São Paulo, Brazil.

[29]. Orleans, L.F. (2007), "ORBITA: Uma Estratégia de Balanceamento de Carga para Tarefas com Restrições Temporais", M.Sc. Dissertation, NCE/UFRJ.

[30] Pradhan, P.; Tewary, R; Sahu, S; Chandra, A. (2002), An observation-based approach towards self managing Web servers. In International Workshop on Quality of Service, Miami Beach, FL.

[31] Schroeder, B.; Harchol-Balter, M. (2006), Achieving class-based QoS for transactional workloads. IEEE International Conference on Data Engineering.

[32] Schroeder, B.; Wierman, A.; Harchol-Balter, M. (2006), Open Versus Closed: A Cautionary Tale. Network System Design and Implementation, San Jose, CA. Pp 239-252.

[33] Serra, A.; Gaïti, D.; Barroso, G.; Boudy, J. (2005), Assuring QoS Differentiation and Load-Balancing on Web Servers Clusters. IEEE Conference on Control Applications, p. 8.85-890.

[34] Wiesmann, M. et al (2000), Understanding Replication in Databases and Distributed Systems, 20th IEEE International Conference on Distributed Computing Systems , p. 464.

[35] Wiesmann, M., Schiper, A. (2005), Comparison of Database Replication Techniques Based on Total Order Broadcast. IEEE Journal Transactions On Knowledge And Data Engineering, v. 17, n. 4, p. 551 – 566.

[36] Wydrowski, B.;Zukerman, M. (2002), QoS in Best-Effort Networks, IEEE Communications Magazine.

[37]. Xiong, M. et al (2002), Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics, IEEE Journal Transactions On Knowledge And Data Engineering, v.14, n.5, p.1155 – 1166

Modelling Reinforcement Learning in Policy-driven Autonomic Management

Raphael M. Bahati and Michael A. Bauer
Department of Computer Science
The University of Western Ontario, London, ON N6A 5B7, CANADA
Email: {rbahati;bauer}@csd.uwo.ca

Abstract

Management of today's systems is becoming increasingly complex due to the heterogeneous nature of the infrastructure under which they operate and what the users of these systems expect. Our interest is in the development of mechanisms for automating the management of such systems to enable efficient operation of systems and the utilization of services. Central to autonomic management is the need for systems to monitor, evaluate, and adapt their own behavior to meet the different, and at times seemingly competing, objectives. Policy-driven management offers significant benefit to this effect since the use of policies can make it more straightforward to define and modify systems behavior at run-time, through policy manipulation, rather than through re-engineering. This work examines the effectiveness of Reinforcement Learning methodologies in determining how to best use a set of active (enabled) policies to meet different performance objectives. We believe that Reinforcement Learning offers significant potential benefits, particularly in the ability to modify existing policies, learn new policies, or even ignore some policies when past experience shows it is prudent to do so. Our work is presented in the context of an adaptive policy-driven autonomic management system. The learning approach is based on the analysis of past experience of the system in the use of policies to dynamically adapt the choice of policy actions for adjusting applications and system tuning parameters in response to policy violations. We illustrate the impact of the adaptation strategies on the behavior of a multi-tiered Web server consisting of Linux, Apache, PHP, and MySQL.

Index Terms—Autonomic Management, Reinforcement Learning, Policy-driven Management, QoS Provisioning.

1 Introduction

Today's Information Technology (IT) infrastructure is becoming heterogeneous and complex to the point that it is extremely difficult, if not impossible, for human operators to effectively manage. Increasingly, the combination of applications integrated within a single or multi-computer environment has become a key component in the way many organizations deliver their services and provide support. Ensuring that such systems meet the expected performance and behavioral needs is among the key challenges facing today's IT community. To this end, there has been a lot of interest in the use of explicit system performance models to capture systems behavior as well as provide guidance in managing applications and systems. While these approaches have achieved some success in specific areas, we note that developing models that accurately capture systems dynamics, particularly for the state of the enterprise systems, is highly nontrivial.

Our interest is in the development of policy-driven autonomic techniques for managing these types of systems. Required or desired behavior of systems and applications can be expressed in terms of policies. Policies can also be used to express possible management actions. As such, policies can be input to or embedded within the autonomic management elements of the system to provide the kinds of directives which an autonomic manager could make use of in order to meet operational requirements. The effective use of policies in autonomic management requires that the policies be captured and translated into actions within the autonomic system. As such, policies can provide the kinds of directives best suited for flexible, adaptive, and portable autonomic management solutions.

Previous work on the use of policies has mainly focused on the specification and use "as is" within systems and where changes to policies are only possible through manual intervention. In an environment where multiple sets of policies may exist, and where at run-time multiple policies may be violated, policy selection is often based on statically configured policy priorities which an administrative user may have to explicitly specify. As systems become more complex, however, relying on humans to encode rational behavior onto policies is definitely not the best way forward. It is imperative, therefore, that autonomic systems have mechanisms for adapting the use of policies in order to deal with

not only the inherent human error, but also the changes in the configuration of the managed environment and the complexities due to unpredictability in workload characteristics.

Self-optimization describes the ability of autonomic systems to evaluate their own behavior and adapt it accordingly to improve performance [1]. In the context where policies are used to drive autonomic management, this may often require having a system monitor its own use of policies to learn which policy actions are most effective in encountered situations. The system might try to correlate management events, actions and outcomes based, for example, on the long-term experience with a set of active policies. This information could then be used to enable the system to learn from past experience, predict future actions and make appropriate trade-offs when selecting policy actions. The use of policies in this context offers significant benefits to autonomic systems in that it allows systems administrators to focus on the specification of the objectives, leaving it to systems to plan how to achieve them. This paper looks at how Reinforcement Learning methodologies could be used to guide this process. In particular, we demonstrate how a model derived from the enabled policies and the consequences of the actions taken by the autonomic system (which we first proposed in [2]) could be "learned" on-line and used to guide the choice of policy actions for adjusting system's tuning parameters in response to policy violations.

The rest of this paper is organized as follows. We begin with a background on Reinforcement Learning in Section 2. In Section 3, we describe the structure of the policies we assume in our work and provide examples illustrating how these policies are used to drive autonomic management. Section 4 presents an adaptive policy-driven autonomic management architecture illustrating key control feedback interactions involved in guiding the selection of policy actions for resolving Quality of Service (QoS) requirements violations. Section 5 and 6 describe how Reinforcement Learning methodologies could be used to model an autonomic computing problem involving QoS provisioning. Section 7 describes the prototype implementation of the learning mechanisms, illustrating the impact of the adaptation strategies on the behavior of a multi-tiered Web server. We review some related work in Section 8, and conclude with a discussion on key challenges and possible direction for future work in Section 9.

2 Reinforcement Learning Background

Reinforcement Learning describes a learning paradigm whereby, through trial-and-error interaction with its environment (see Figure 1), an agent learns how to best map situations to actions so as to maximize long-term benefit [3]. As such, Reinforcement Learning is often associated with training by reward and punishment whereby, for each action the agent chooses, a numeric reward is generated which indicates the desirability of the agent being in a particular state. A key distinction between Reinforcement Learning and other forms of learning is on what information is communicated to the learner after an action has been selected. In *supervised learning*, for example, the learner only has to visit a state once to know how to act optimally if it encounters the same state again. This is because, for each action taken, the learner is told what the correct action should have been. In Reinforcement Learning, on the other hand, the learner only receives a numeric reward which indicates how good the action was (as opposed to whether the action was the best in that situation). The only way for the learner to maximize this reward, therefore, is to discover which actions generate the most reward in a given state by trying them. Consequently, the learner is often faced with a dilemma: whether to use its current knowledge to select the best action to take, (*exploit*) or try actions it has not yet tried (*explore*) in order to improve its guesses in the future.

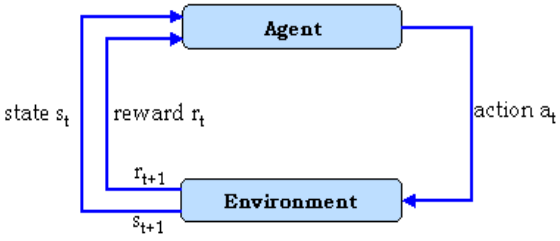


Figure 1. The agent-environment interaction in Reinforcement Learning [3].

As with many learning problems, it is often impractical to obtain an environment model that is both accurate and representative of all possible situations the learning agent may encounter while interacting with the environment [3]. While model-free Reinforcement Learning methods exist which are guaranteed to find optimal policies (i.e., choices of actions per situation), they make extremely inefficient use of data they gather [4]. One approach for overcoming this shortfall is for the agent to learn the model of the environment's dynamics, on-line, as it interacts with the environment. This has been demonstrated to significantly accelerate the learning process (see, for example, [5, 6, 7]). In this approach, a model is updated continually throughout the agent's lifetime: at each time step, the currently learned model is used for planning; i.e., using the learned model to improve the policy guiding the agent's interaction with the environment.

Several model-based learning algorithms exist in the literature and differ mainly on how the model *updates* are per-

Algorithm 1 Dyna-Q

Input: Initialize Model(s, a) for all $s \in S$ and $a \in A(s)$

1: **for** $i = 1$ *to* ∞ **do**
2: $s \leftarrow$ current (non terminal) state
3: $a \leftarrow \epsilon$-greedy(s, Q)
4: Execute a; observe resultant state, s', and reward r
5: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
6: Model$(s, a) \leftarrow s', r$
7: **for** $j = 1$ *to* k **do**
8: $s \leftarrow$ random previously observed state
9: $a \leftarrow$ random action previously taken in s
10: $s', r \leftarrow$ Model(s, a)
11: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
12: **end for**
13: **end for**

formed. In this paper, we make use of an algorithm called Dyna-Q [8] (see Algorithm 1) which estimates action-values; i.e., a measure of how good it is for an agent to perform a particular action in a given situation. Briefly, the algorithm works as follows: Beginning with state s, the agent selects action $a \in A(s)$ and observes the resultant state s' and reward r. Using this information, the agent updates the action-value associated with action a (line 5) and adds this information to the current system model (line 6). It also performs k additional updates of the model by randomly selecting and updating the action-value estimates of k state-action pairs (lines 7 - 12). In the sections that follow, we describe how *model-learning* mechanisms could be applied to an autonomic computing problem involving QoS provisioning. But first, we begin with a look at how policies could be used to drive autonomic management.

3 Autonomic Management Policies

We have been exploring the use of policies as the basis for autonomic management, with a particular focus on e-commerce systems. We feel that policies can provide the kinds of directives which autonomic systems can and should rely on when making management decisions. As with much of the previous work on policy-driven management (see, for example, [9, 10]), our interest is on *action policies* (expressed as *obligation policies* in Ponder [9]) since they can be defined and modified on a per component basis and can provide useful information for autonomic managers. The use of action policies within autonomic computing is likely to continue partly due to their simplicity and, unlike *goal policies* [11, 12] and *utility policies* [11, 13, 14, 15], do not require a system model in order to be used [11]. In this work it is assumed that action policies are event-triggered, action-condition rules [9]. An event triggers the evaluation of a rule of the form "**if** [conditions] **then** [actions]". An event is generated as a result of some condition of the state of the system being true. This section looks at what these policies are and how they could be used within autonomic computing.

3.1 Policy Structure

We assume a policy to consist of several attributes including one or more conditions and an ordered list of actions that make adjustments to some tuning parameters:

3.1.1 Policy Rule

A policy rule basically consists of a *policy type* (discussed in Section 3.2), *policy name*, a *conditions set* which is dependent on one or more conditions, and an *actions set* (see, for example, Figure 3). Because a policy may apply to many different components, the assumption is that the policy would be instantiated at run-time, say when the management system starts its components or a particular application is started. For example, the policy *target* might be instantiated to a particular host within a network of hosts, that is, the same policy could apply to each of the hosts though each would be monitored separately. In the policy example of Figure 3, the policy target is the process corresponding to the Policy Enforcement Point (PEP). The policy *subject* is the management component that should receive the event when there is a violation. Hence, the subject would also be instantiated. In our prototype, the subject of an *expectation policy* (see Section 3.2.2) would likely be the process corresponding to the Policy Decision Point (PDP) as illustrated by the policy of Figure 3; in larger systems, there could be other management components to receive events or multiple PDPs. A policy has at least one other attribute which can change dynamically, which specifies whether a policy is enabled (set to true) or not.

3.1.2 Policy Condition

A policy condition captures the state of an application, a system, device, etc. It is assumed that events are generated from monitoring components and that the Event Handler (see Section 4.1) filters received events for those of "interest". An event specified by name only is essentially a Boolean value; i.e., the occurrence of the event itself is sufficient to take an action. An event with an attribute indicates that the value of the attribute is to be used in evaluating an expression, such as comparing the value to a threshold. It is also possible to have a policy which becomes violated only when multiple events or conditions occur. These are specified via the standard logical operators.

```
configuration policy{InstallCPUMonitor(MonitorManager,localhost)}
 if(INSTALL:CPUMonitor = true)
 then{./CPUMonitor test{IsConditionEnabled(CPU:utilization) = true}}
```

Figure 2. A configuration policy for installing a CPU Monitor.

3.1.3 Policy Action

A policy action defines what has to be executed should the condition(s) specified in the policy hold true. Each action is, essentially, the name of a function that should be executed. The function may have parameters that would be determined from the information associated with the policy, e.g., domain, events, event attributes, etc. One or more actions may also be specified. Each action may have an optional `test` associated with it, with or without parameters. The test can be used to determine if the component state or context invalidates the particular action. Such a test is a Boolean function or could return a value which is then compared to some threshold value. If the result of the test expression is "true" then that indicates that the action is enforceable; note that the negation of a test is permitted in which case the expression is "true" if the test evaluates to "false". As such, policy tests provide ways in which the degree of self-management could be controlled. Action sequences may be conjoined (i.e., "AND-ed" together) indicating that all the actions in the sequence should be executed. Alternative action sequences may also be specified in which case only one of the elements of the sequences would be selected.

In our current approach, we permit only a single action within a single expectation policy to be executed. This is done for two reasons. First, this is a strategy of "doing something simple" and seeing if there is a positive effect. If the change is not sufficient, then a violation is likely to occur again and a further action (which could be the same, e.g., increasing or decreasing the value of a parameter) can be taken. The management cycle in the implementation is short enough that this can happen quickly. Second, taking multiple actions makes it difficult to understand the impact of the actions; e.g., were they all necessary, were some more effective than others, etc. By having the autonomic manager take a single action and log that action and other information, an analysis component can examine that information and possibly determine which action(s), or the order thereof, is better, etc. We outline one such an approach in Section 6.

3.2 Policy Types

We are currently exploring the use of several types of policies for driving autonomic management.

3.2.1 Configuration Policies

Configuration policies describe those policies that are used to specify how to configure and install applications and services. This may include, for example, setting static configuration parameters based on the Service Level Agreement (SLA) requirements (e.g., performance, availability, quality of service), the given or expected environmental parameters (e.g., required services, number of active users), and the available resources (e.g., number of processors, processor speed, memory size, disk space).

A sample configuration policy for installing a CPU Monitor for the system of Figure 4 is shown in Figure 2. In this example, the `MonitorManager` (i.e., the poliy subject) is the component responsible for installing the CPU monitor on a `localhost` (i.e., the policy target). The policy test determines the conditions under which the CPU Monitor is to be installed. In this example, the monitor is installed only if the condition "`CPU:utilization`" is enabled - as determined by a set of enabled *expectation policies* (see Section 3.2.2). As such, changes to the policies driving autonomic management could also trigger dynamic reconfiguration of systems and applications. For example, by disabling the policy of Figure 9 (assuming, of course, that it is currently the only policy with a "`CPU:utilization`" condition), the CPU Monitor would be disabled as a result. A key advantage here is the reduction in the management overhead since events specific to CPU utilization would no longer be relevant when the policy is no longer active.

3.2.2 Expectation Policies

Expectation policies define information used to ensure that operational requirements are met and expected conditions not violated. We have also been using expectation policies to indicate how the system could optimize its use of resources. For example, a policy could indicate that, when the response time of requests to the server falls below a certain level, then Apache processes handling requests could be reduced. This would then free up system resources.

A sample expectation policy for resolving violations in Apache's response time is shown in Figure 3. It consists of two conjunctive conditions and three disjunctive actions. Note that the actions, which specify adjustments to the application's tuning parameters, are quite simple since each specifies a small - and in some cases the smallest pos-

```
expectation policy{RESPONSETIMEViolation(PDP, PEP)}
 if(APACHE:responseTime > 2000.0) & (APACHE:responseTimeTREND > 0.0)
 then{AdjustMaxClients(+25) test{newMaxClients < 151} |
      AdjustMaxKeepAliveRequests(-30) test{newMaxKeepAliveRequests > 1} |
      AdjustMaxBandwidth(-128) test{newMaxBandwidth > 255}}
```

Figure 3. A sample expectation policy for resolving Apache's response time violation.

sible - increment/decrement in the value of the parameter. For example, the Apache's `MaxClients` parameter could only be adjusted in increments/decrements of the number of threads per child process, as specified in the server's configuration. Thus, a general knowledge of how an increase/decrease in the value of a particular parameter impacts a system's performance metrics may be sufficient to define reasonable policies. For instance, a violation in Apache's response time could be due to the fact that there aren't enough server processes to handle clients requests, in which case increasing `MaxClients` could resolve the problem. If this is no longer possible (as determined by the action test), one might try to reduce the amount of time (i.e., `MaxKeepAliveRequests`) existing clients hold onto the server processes. And, if this is no longer possible, it could be that the server is overwhelmed by the number of requests in which throttling some may alleviate the problem. This is illustrated by the expectation policy in Figure 3. Since, only a single action could be executed, the order in which the actions are specified within the policy is also important. In this case, more drastic actions could be taken once it is no longer possible, for example, to meet the objectives through tuning application's parameters. This is precisely the purpose of the action "`AdjustMaxBandwidth(-128)`" which throttles requests to the server by reducing the rate at which the server processes clients requests based on a client's service class (see Section 7.3 for details).

3.2.3 Management Policies

Management policies deal with information and actions for managing the management system itself or for the overall administration of the system or applications. Such policies may include those for the prioritization of expectation policies, for diagnosis in determining an action, or involving some analysis (say of previous behavior) in determining an action. We are currently exploring the use of management policies to guide the on-line learning process. Our particular focus is on how the learning algorithms could be optimized to be less computational intensive in order to meet the resource constraints imposed by the environment. We comment further on this in Section 9.

4 System Architecture

A detailed view of the architecture for the adaptive policy-driven autonomic management system is depicted in Figure 4. Our approach to autonomic management involves providing quality of service support local to each host. Each local host, therefore, has a single Policy Decision Point (PDP) whose responsibility is to oversee the management of a single host according to the policies specified. In a multi-tiered Web-server environment, for example, several components (i.e., a Web server, an application server, and a database server) may cooperate to deliver a set of services. In the case that all these components are run on a single host, a local PDP will be responsible for ensuring that the managed application, as a whole, behaves as expected. Since each component would have it's own set of policies, more complex decisions regarding the choices of actions when multiple policies, possibly from multiple components, are violated will be confined to a single Event Analyzer. In the case where each component of the multi-tiered Web server is run on a different host, several PDPs could be configured to oversee the management of each local host where the individual component is run. However, a single Event Analyzer is used to co-ordinate the activities of the individual PDPs on each local host in order to provide quality of service support spanning multiple hosts. A key advantage of a de-centralized approach to QoS support is that the autonomic system is likely to be more scalable and responsive since QoS decisions specific to local behavior will be confined locally [16]. By reducing the distance between the *autonomous management system* and the *managed system* (see Figure 5), less overhead is incurred, in part, as a result of using more efficient local communication mechanisms between components [16]. In this section, we highlight key functionality of the different components.

4.1 Architectural Components

The following are the key components of the architecture for the adaptive policy-driven autonomic management:

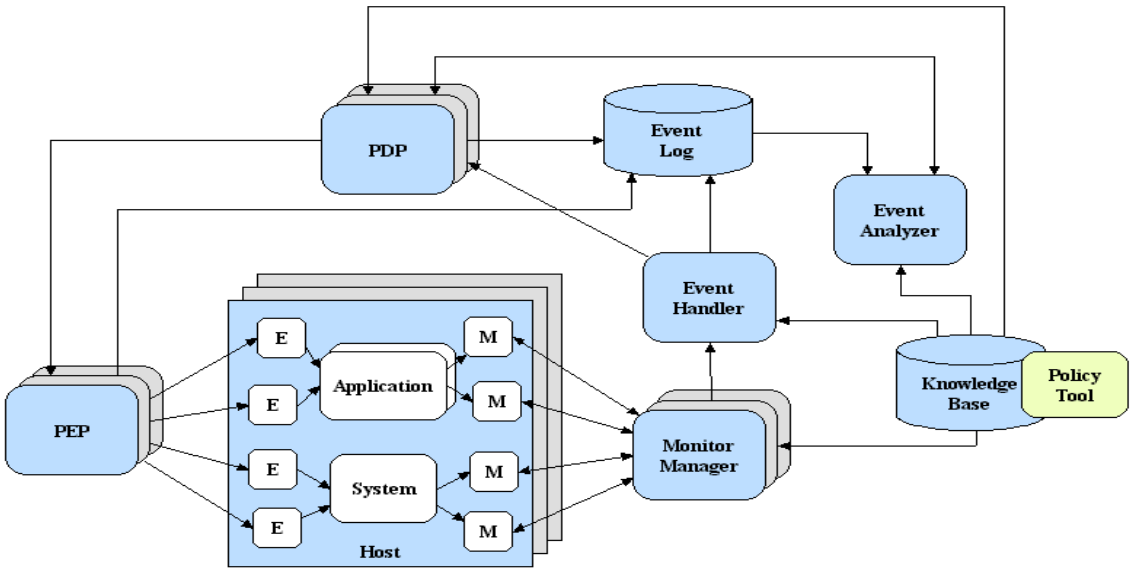


Figure 4. The adaptive policy-driven autonomic management architecture.

4.1.1 Knowledge Base

The Knowledge Base is a shared repository for system policies and other relevant information. This may include information for determining corrective actions for resolving QoS requirements violations as well as configuring systems and applications. The information about policies is eventually distributed to other management components, and then realized as actions driving the autonomic management.

4.1.2 Monitor (M)

Monitors gather performance metric information of interest for the management system such as resource utilization, response time, throughput and other relevant information. It is this information that is then used to determine whether the QoS requirements are either being met or violated.

4.1.3 Monitor Manager

Monitor Manager deals with the management of Monitors, including instantiating (i.e., loading and starting) a Monitor for a certain resource type to be monitored as well as providing the context of monitoring (i.e., monitoring frequency or time interval for periodic monitoring or monitoring times for scheduled monitoring). In addition, it allows Monitors to be re-configured (i.e., adding a new Monitor, adjusting the context of monitoring, or disabling a Monitor) dynamically in response to run-time changes to policies. At the core of its responsibility is the collection and processing of Monitor events whose details are then reported to the Event Handler. In essence, the Monitor Manager acts as an *event producer* by gathering information from multiple Monitors as illustrated in Figure 4. It provides customized services to *event consumers* (such as the Event Handler) in terms of how often they should receive events notifications.

4.1.4 Event Handler

The Event Handler deals with the processing of events from the Monitor Manager to determine whether there are any QoS requirements violations (based on the enabled policy conditions) and forwarding appropriate notifications to the interested components. This includes notifying the PDP of conditions violations as well as forwarding information to the Event Log for archiving. A key feature of this component is its ability to provide customized services to event consumers (i.e., PDP, Event Log, etc.) through subscriptions by allowing components to specify, for example, how often and/or when they should receive notifications.

4.1.5 Policy Decision Point (PDP)

This component is responsible for deciding on what actions to take given one or more violation messages from the Event Handler. The PDP must decide which policy, if any expectation policy has been violated, was the “most important” and then what action(s) to take. It uses information not only about the violations, but also the expectation policies and management policies, both expressed within the expectation policies and via management policy rules.

4.1.6 Policy Enforcement Point (PEP)

This component defines an Application Programming Interface (API) which maps the actions subscribed by the PDP to the executable elements; i.e., the various Effectors.

4.1.7 Effector (E)

Effectors translate the policy decisions, i.e., corrective actions, into adjustment of configuration parameters to implement the corrective actions. Note that there will be multiple instances of the Effectors for different types of resources (e.g., logical partitioning of CPUs, allocation of streaming buffers) or tuning parameters to be adjusted.

4.1.8 Event Log

This component archives traces of the management system's events onto (1) an event log in the memory for capturing recent short term events, and (2) a persistent event log on disk for capturing long term history events for later examination. Such events may include QoS requirements violations from the Event Handler, records of decisions made by the PDP in response to the violations, the actions enforced by the PEP, as well as other relevant management events.

4.1.9 Event Analyzer

This component correlates the events with respect to the contexts, performs trend analysis based on the statistical information, and models complex situations for causality analysis and predictive outcomes of corrective actions, to enable the PDP to learn from past, predict future and make appropriate trade-offs and optimal corrective actions.

4.2 Component Interaction

Figure 5 illustrates key interactions driving autonomic management. In this approach, we make use of policies to specify both the expected performance behavior of the managed systems as well as decisions driving autonomic management. Such policies are specified via the Policy Tool (see Figure 11). The management system can also adapt, dynamically, to handle changes to policies made via the interface. This approach is illustrated in the diagram and is characterized by the interaction between the *Managed System* and the *Autonomous Management System*. In essence, the management system determines what to monitor based on the policies that are active and determines if changes should be made. Any changes are done through effectors which can change the values of various parameters of the applications (e.g. Apache or other components) or change the operation of the system itself, such as blocking requests or adding/removing processes. This section looks at how the different components interact to achieve the different performance objectives in the context of self-configuration and self-optimization.

4.2.1 Self Configuration

Briefly, the management system in Figure 4 is instantiated by first invoking the Management Agent (not shown in the diagram). The initial task of this agent is to query all the enabled *configuration policies* (see Section 3.2.1) from the policy repository. It is these policies that are used to install the management components, with the exception of Monitors, the responsibility of which falls to the Monitor Manager; i.e., the policy `subject` (see Figure 2). The PDP, in turn, queries the policy repository for all the enabled *expectation policies* (see Section 3.2.2) and uses this information to make decisions on how to respond to violations. Once the different management components have been installed, the manager's responsibility becomes ensuring that appropriate components are notified if there are any changes to the policies governing the behavior of the system.

To illustrate the impact of disabling a policy, let's assume that the policies of Figures 3 and 9 are the only enabled expectation policies and a user disables the latter policy. This would trigger four specific notifications: (i) The first notification would be forwarded to the PDP since this component is the subject of the policy. The PDP in turn would update its policies accordingly, i.e., by removing the CPU violation expectation policy. (ii) The second notification would be forwarded to the Event Handler, the component responsible for determining whether the QoS requirements are being met. Disabling the policy of Figure 9 means that the conditions "`CPU:utilization`" and "`CPU:utilizationTREND`" must also be disabled. This would prevent any notifications from being forwarded to the PDP should a violation of any of the conditions occur. (iii) The third notification would be forwarded to the Monitor Manager to determine whether any of its Monitors are to be disabled as a result. In this particular case, the CPU Monitor would be disabled since events specific to CPU utilization are no longer relevant. This directive is captured by the test "`IsConditionEnabled(CPU:utilization)`" as part of the action to install the CPU Monitor (see the policy of Figure 2). (iv) The fourth and final notification would be forwarded to the Event Analyzer, which may need to update policy state information since the change may affect the learning process. This type of adaptation is the focus of our current research and will not be addressed here.

4.2.2 Self Optimization

Self-optimization deals with adapting the behavior of applications as well as systems in order to meet specific performance objectives. In the context of where policies are used

to drive autonomic management, adaptation may be specific to the choice of policy actions. Figure 5, in particular, illustrates two main feedback control loops that drive how the autonomic system adapts the way it responds to the violations in the QoS requirements of the managed system.

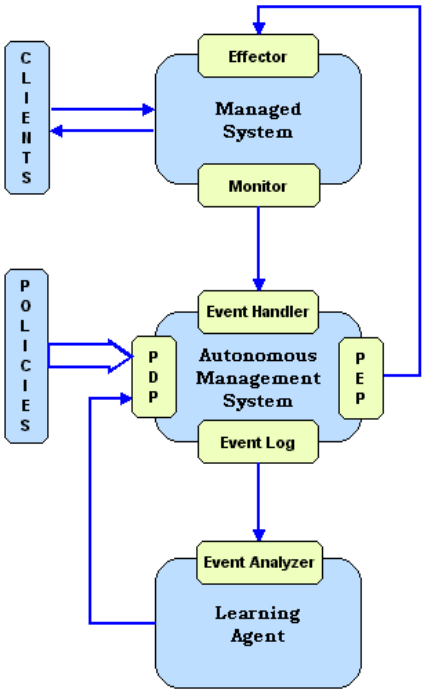


Figure 5. Two feedback loops driving the autonomic management of a managed system.

The first control loop, which constitutes a single management cycle, consists of monitoring the behavior of the managed system and, using the information collected within the interval, selecting policy actions to resolve violations.

1. The Monitors collect and forward performance metric information to the Monitor Manager (not shown in the diagram) which is then processed (i.e., for averages and trends) and forwarded to the Event Handler.

2. The Event Handler's responsibility is to determine whether the QoS requirements of the managed system have been violated. For each violation, a notification is forwarded to the PDP.

3. For each management interval, the PDP collects all the violation messages and processes them to determine whether any of the enabled expectation policies has been violated. The PDP then determines the order in which the actions advocated by the violated policies are to be "tried" (based on the violation information collected during the interval). The ordered actions are then forwarded to the PEP.

4. On receiving the policy actions, the PEP performs tests associated with each action, and if successful, invokes the appropriate Effector(s) to perform the actual adjustment to the managed system's parameter(s). Note that, in our current implementation, we only permit a single action to be executed - for the reasons discussed in Section 3.1.3.

The above control mechanisms were the focus of our initial investigation on the performance behavior of the Apache Web Server (see, for example, [17]). This work was later extended to incorporate adaptation strategies on how the PDP selects policy actions from those advocated by the violated policies in the context of a multi-component Web server (see [18]).

The second feedback loop deals with self-optimization; i.e., the ability of systems to evaluate their own behavior and adapt it accordingly to improve performance. In the context of where policies are used to drive autonomic management, this often requires monitoring the behavior in the use of policies and using the experience to learn optimal policies; i.e., the selection of optimal policy actions for each encountered situation. The use of policies in this context offers significant benefits to autonomic systems in that it allows systems administrators to focus on the specification of the objectives leaving it to systems to plan how to achieve them. The key steps of the feedback loop include the following:

1. Process the Event Log information (which includes Monitor events, QoS requirements violation events, decisions made by the PDP in response to the violations, and the actions enforced by the PEP), on-line, to model the performance of the managed system based on the observed experience in the use of policies.

2. Use the model, when possible, to advise the PDP on how to adapt its action selection mechanisms based on the current state of the system.

Figure 6 summarizes the key interactions between the different components involved in coordinating the steps of the two feedback control loops during a single management cycle. The Policy Tool (see Figure 11), in this case, provides an interface to the autonomous management system through which users can manage (i.e., add, modify, delete) policies governing the behavior of the system.

1. **Monitors:** Collect performance metric information of interest from the managed environment (E) and forward it to the Monitor Manager.

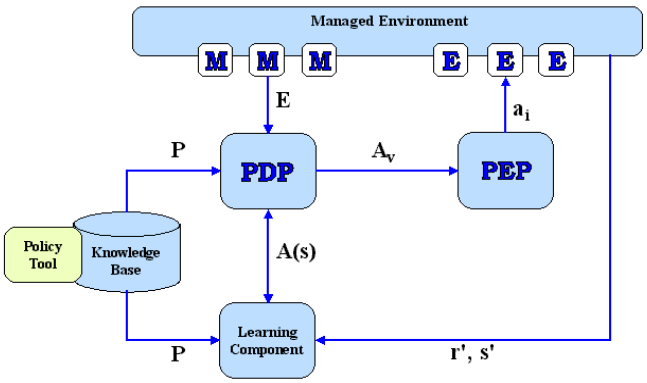


Figure 6. Feedback control interactions.

2. **Monitor Manager:** Process the Monitor events for averages and trends and forward the processed information to the Event Handler.

3. **Event Handler:** Determine whether any QoS requirements have been violated. For each violation, forward a notification, e_i, to the PDP.

4. **PDP:** During each management interval, form a set, P_v, of violated policies (from the enabled policies set P) based on the violation notification events, $e_i \in E_v$, received during the interval. A policy is said to be violated if all its conditions evaluate to true when matched against violation events in E_v.

5. **PDP:** Decide whether to use the knowledge learned from past experience with a set of active policies, P, to select the best action to take (i.e., by requesting advise from the learning component with probability $1 - \epsilon$), or to try actions not yet tried (with probability ϵ).

6. **PDP:** Form set A_v corresponding to the actions associated with the current state, $A(s)$, if the state has previously been encountered, then continue with 10 (exploit); Otherwise, continue with 7 (explore).

7. **PDP:** Compute the severity of each condition in P_v using the values of the violation events in E_v.

8. **PDP:** Form a set, A_v, of unique policy actions based on the actions advocated by the violated policies in P_v.

9. **PDP:** Compute $Q_0(s, a)$ for each policy action in A_v. $Q_0(s, a)$ estimates the initial action-value of the policy actions based on the characteristics of both violation events and the enabled policies (see Equation 6).

10. **PDP:** Sort the actions in A_v by the action-value estimate, $Q(s, a)$, and forward them to the PEP. The aim here is to ensure that actions with the highest value are tried first. Since only a single action is executed, the order in which the actions are arranged is of great importance.

11. **PEP:** Validate the policy actions in A_v by performing the tests associated with each action (see, for example, Figure 3) and then invoke the appropriate Effector (E) to perform the actual action, a_i, for the first action to pass the tests.

12. **Learning Component:** Observes the resultant state s' and reward r'. Using the Dyna-Q algorithm (see Algorithm 2), update the current system model.

In the next Section, we elaborate on how the above feedback control interactions are modelled onto a Reinforcement Learning problem.

5 Modelling Reinforcement Learning

A model, in Reinforcement Learning, describes any feedback that guides the interaction between the learning agent and its environment. This interaction is driven by the choices of actions and the behavior of the system as a consequence of taking those actions. In the context of a policy-driven autonomic management agent, the choices of actions are determined by the expectation policies that are violated. This section looks at what constitutes a state-transition model and how this structure is derived. We first begin by formally defining expectation policies.

Definition 1 *An expectation policy is defined by the tuple* $p_i = \langle C, A \rangle$ *where:*

- C is conjunctive conditions associated with policy p_i with each condition, $c_j \in C$, defined by the tuple $c_j = \langle$`ID, metricName, operator,` $\Gamma\rangle$, where; `ID` is a unique identification for the condition; `metricName` is the name of the metric associated with the condition; `operator` is the relational operator associated with the condition; and Γ is the threshold of the condition.

- A is a set of actions associated with policy p_i with each action, $a_j \in A$, defined by the tuple $a_j = \langle$`ID, function, parameters,` $\tau\rangle$, where; `ID` is a unique identification for the action; `function` is the name of the function (within the PEP) that should be executed; `parameters` is a set of function parameters; and τ is a set of tests associated with the action.

An expectation policy condition essentially identifies the region (or interval) on the side of the condition's threshold (based on the condition's operator) where a condition is said

to be violated. Thus, our expectation policy conditions only consider "$>$, $\geq$, $<$, and $\leq$" operators. For a policy condition "`APACHE:responseTime > 2000.0`", for example, any response time measurement beyond 2000.0 ms would be considered as a violation, the severity of which increases the further the measurement is from the threshold. In our implementation of expectation policies, it is assumed that the quality of service specific to a metric's measurement deteriorates, i.e., monotonically decreases, as the measured value increases. In essence, the main objective for the autonomic manager is to steer the system towards metrics' regions where the quality of service is the highest; i.e., towards the most desirable regions.

A policy-driven autonomic management system is likely to consist of multiple expectation policies, a subset of which may be active (or enabled) at any given time; which brings us to our next definition.

Definition 2 *Suppose that P^A denotes a set of all expectation policies such that $p_i \in P^A$ where $p_i = \langle C, A \rangle$. Let P be a subset of expectation policies an autonomic manager uses to make management decisions; i.e., $P \subseteq P^A$. A policy system corresponding to P is defined by the tuple $PS = \langle P, W_C \rangle$ where:*

- $W_C = \langle c_i, \omega_i \rangle$ associates each policy condition, c_i, with a weight, ω_i, such that, for all $c_i \in p_m$ and $c_j \in p_n$, $\omega_i = \omega_j$ if $c_i = c_j$.

The conditions' weights, which are specified manually in our current implementation, provide a way of distinguishing policy conditions based on the significance of violating a particular metric. In essence, W_C provides a way of biasing how the autonomic system responds to violations; we elaborate further on this in Section 6.1.

To model system's dynamics from the use of an active set of policies, we make use of a mapping between the enabled expectation policies and the managed system's states whose structure is derived from the metrics associated with the enabled policy conditions.

Definition 3 *A policy system $PS = \langle P, W_C \rangle$ derives a set of system metrics, $m_i \in M$, such that, for each $C \in p_j$ where $p_j \in P$, $M = \bigcup_{c_i \in C} \{c_i.\texttt{metricName}\}$.*

In this approach, a *state-transition model* (see Definition 4) is defined which uses a set of active expectation policies (see, for example, Figure 3) to create a set of policy-states and the actions of the management system to determine transitions between those states. This mapping is motivated by two key observations about the expectation policies. First, they define what the expected performance and behavioral objectives are (as captured by the conditions of the enabled expectation polices). Second, they define the choices of actions whenever the specified objectives are violated. In essence, the interaction between the learning agent and its environment is driven, partly, by the enabled expectation policies. Thus, we assume a standard Markov Decision Process (MDP) [3, 4].

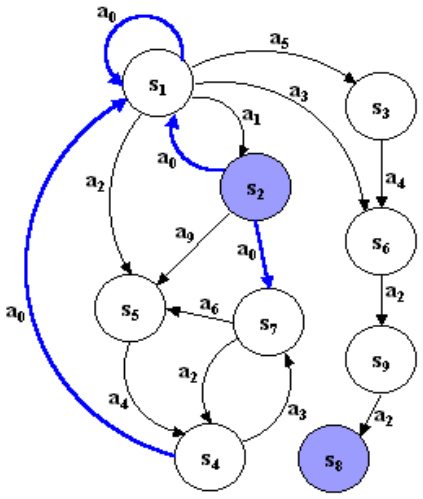


Figure 7. A sample state transition graph.

Definition 4 *A state-transition model derived from the policy system $PS = \langle P, W_C \rangle$ is defined by the graph $G^P = \langle S, T \rangle$ where:*

- S is a set of system states (see Section 5.1.) derived from the metrics of the conditions of the enabled expectation policies.
- T is a set of transitions (see Section 5.2) where each transition, $t_i \in T$, corresponds to a directed edge on the graph. A transition is determined when the autonomic manager takes an action as a result of being in one state, which may, or may not, result in a transition to another state.

As such, we capture the management system's behavior in the use of an active set of policies using a state-transition graph. This is illustrated in Figure 7 which shows the different types of states (see Definition 7) as well as transitions between states as a result of either the actions of the autonomic manager (i.e., a_i) or other dynamic characteristics outside the control of the autonomic manager (i.e., a_0); we elaborate further on this in Section 5.2. The result can then be used by the autonomic manager to consider choices of policy actions that it might take when it determines that the system is in a particular state. That is, the autonomic manager could take an action as defined below:

$$a_i \in A(s_i) \tag{1}$$

where $A(s_i)$ is a set of actions advocated by the expectation policies that are violated when the system is in state s_i. The information about the system that is used to determine the state-transition graph is extracted from the Event Log as illustrated in Figure 5. For a given set of active policies, this structure is built dynamically as the events from the different management components are recorded in the logfile. Note that, since the autonomic manager records only those states that are experienced, "states explosion" is likely to be restricted. We comment further on this in Section 9.

5.1 System States

As indicated, states are based upon the metrics in the conditions of the policy system. We define states through the following definitions.

Definition 5 *A policy system $PS = \langle P, W_C \rangle$ with metrics set M derives a set of metric-regions, M_R, for each metric $m_i \in M$, $r_{m_i} \in M_R$, whose structure is defined by the tuple $r_{m_i} = \langle \alpha_{m_i}, \sigma_{m_i} \rangle$, where:*

- $\alpha_{m_i} = \langle \texttt{ID}, \texttt{metricName}, \omega \rangle$ corresponds to a unique metric from among the metrics of the conditions of the policies in P; such that, `metricName` is the name of the metric and ω is the weight of the condition (see Definition 2) associated with metric m_i. In the case that a single metric is associated with more than one policy condition and where each condition might have different weights, the value $m_i.\omega$ is computed as follows:
$$m_i.\omega \;=\; \max_{c.m_i \in C} c.\omega \qquad (2)$$
which essentially corresponds to the weight of the condition with the largest weight value from among the conditions associated with metric m_i.

- $\sigma_{m_i} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_k\}$ is a set of thresholds from the conditions associated with metric m_i such that, $\Gamma_i < \Gamma_j$ if $i < j$. As such, σ_{m_i} derives a set of metric regions which map the observed metric measurement onto appropriate localities (i.e., intervals) as defined by the thresholds of the policy conditions associated with metric m_i, such that $R_{m_i} = \{R^1_{m_i}, R^2_{m_i}, \ldots, R^{k+1}_{m_i}\}$, where $R^1_{m_i} = (-\infty, \Gamma_1)$; $R^2_{m_i} = (\Gamma_1, \Gamma_2)$; and, $R^{k+1}_{m_i} = (\Gamma_k, \infty)$.

Thus, if $\sigma_{m_i} = \langle \Gamma_1, \Gamma_2 \rangle$, for example, it would yield three regions in our approach: $R^1_{m_i} = (-\infty, \Gamma_1)$, $R^2_{m_i} = (\Gamma_1, \Gamma_2)$, and $R^3_{m_i} = (\Gamma_2, \infty)$; which brings us to our next definition.

Definition 6 *Given a set of metric-regions for each metric $m_i \in M$, $r_{m_i} \in M_R$, such that $r_{m_i} = \langle \alpha_{m_i}, \sigma_{m_i} \rangle$, where σ_{m_i} derives a set of metric regions $R^j_{m_i} \in R_{m_i}$; we define a mapping function, $f(R^j_{m_i}) \rightarrow \mathbb{R}$, which assigns a numeric value to the j-th region in R_{m_i} such that, $f(R^k_{m_i}) > f(R^l_{m_i})$ if $k < l$.*

An example of such a mapping, which we make use of in our current implementation, is defined by Equation 3:

$$f(R^j_{m_i}) \;=\; 100 - (\frac{100}{n-1})(j-1) \qquad (3)$$

where n is the total number of regions in R_{m_i}. This function assigns a numeric value between 100 and 0 for each metric's region in R_{m_i}, starting from 100 for the most desirable region and decrementing at equal intervals towards the opposite end of the spectrum, whose region is assigned a value of 0. This approach guarantees that the highest value is assigned to the most desirable region (i.e., the region corresponding to the highest quality of service), assuming, of course, that the assumptions about the conditions of the expectation policies hold (see Definition 1).

Definition 7 *A policy system $PS = \langle P, W_C \rangle$ with metrics M and metrics-regions M_R derives a set of system states S such that, each state $s_i \in S$ is defined by the tuple $s_i = \langle \mu, M(s_i), A(s_i) \rangle$, and where:*

- μ is a `type` which classifies a state as either "violation" or "acceptable" depending, respectively, on whether or not there are any policy violations as a result of visiting a particular state. As noted previously, a policy is said to be violated if all its conditions evaluate to true when matched against violation notifications received during a single management cycle.

- $A(s_i)$ is a set of actions advocated by the expectation policies in P that are violated when the system is in state s_i.

- $M(s_i)$ is a set of state metrics for each metric $m_j \in M$, $r_{m_j} \in M_R$, $r_{m_j} = \langle \alpha_{m_j}, \sigma_{m_j} \rangle$, such that each state metric $s_i.m_j \in M(s_i)$ is defined as follows:

Definition 8 *A state metric $s_i.m_j \in M(s_i)$ given $\alpha_{m_j} = \langle \texttt{ID}, \texttt{metricName}, \omega \rangle$ and $\sigma_{m_j} = \langle \Gamma_1, \Gamma_2, \ldots, \Gamma_k \rangle$ is defined by the tuple $s_i.m_j = \langle \texttt{ID}, \omega, \texttt{value}, R^l_{m_j} \rangle$ where:*

- `ID` is an integer value that uniquely identify each metric $m_i \in M$.

- ω is the weight associated with metric m_i.

- `value` is the observed metric measurement, or average value when state s is visited multiple times.

m_i	c_k	Policy Condition	R_{m_i}	$R^j_{m_i}$	$f(R^j_{m_i})$
m_1	c_1	APACHE:responseTime > 2000.0	$m_1.value \leq 2000.0$ $m_1.value > 2000.0$	$R^1_{m_1}$ $R^2_{m_1}$	100 0
m_2	c_2	APACHE:responseTimeTREND > 0.0	$m_2.value \leq 0.0$ $m_2.value > 0.0$	$R^1_{m_2}$ $R^2_{m_2}$	100 0

Table 1. A metrics structure derived from the policy system of Example 1.

State	$R^k_{m_j}$		$f(R^k_{m_j})$		$A(s_i)$	
s_i	$R^k_{m_1}$	$R^k_{m_2}$	$f(R^k_{m_1})$	$f(R^k_{m_2})$	a_l	State action
s_1	$R^2_{m_1}$	$R^2_{m_2}$	0	0	a_0	γ-action
					a_1	AdjustMaxClients(+25)
					a_2	AdjustMaxKeepAliveRequests(-30)
					a_3	AdjustMaxBandwidth(-128)
s_2	$R^2_{m_1}$	$R^1_{m_2}$	0	100	a_0	γ-action
s_3	$R^1_{m_1}$	$R^2_{m_2}$	100	0	a_0	γ-action
s_4	$R^1_{m_1}$	$R^1_{m_2}$	100	100	a_0	γ-action

Table 2. Sample policy states based on the metrics structure of Table 1.

- $R^l_{m_j}$ is the region corresponding to a region in σ_{m_j} in which the average metric measurement (i.e., value) falls; i.e., if $R^l_{m_j} = (\Gamma_1, \Gamma_2)$, then $\Gamma_1 < \texttt{value} < \Gamma_2$. For each such region, $f(R^l_{m_j})$ then associates a value as described by Equation 3.

Using this approach, each state can be uniquely identified by the region occupied by each state metric based on the conditions of the expectation policies and the value associated with each metric. That is, for a set of policies involving n metrics, each state would have n metrics $\{m_1, m_2, ..., m_n\}$ and, for each metric a specific region whose intervals are derived from the thresholds of the conditions associated with the metric. To elaborate this further, consider the following examples:

Example 1 *Suppose that, policy system $PS = \langle P, W_C \rangle$ currently consists of a single active (enabled) expectation policy shown in Figure 3 (i.e., p_1) such that $P = \{p_1\}$.*

From the conditions of the policy, states derived from the policy system of Example 1 would consist of two metrics; i.e., $M = \{m_1, m_2\}$ where m_1 ="APACHE:responseTime" and m_2 ="APACHE:responseTimeTREND". It follows from Definition 5 that $\sigma_{m_1} = \{2000.0\}$ and $\sigma_{m_2} = \{0.0\}$. As such, metric m_1 would map onto two regions; the response time is either greater than 2000.0 or not. Similarly, metric m_2 would map onto two regions; the response time trend is either greater than 0.0 or not. This is illustrated by the regions shown in Table 1. In the case of the two regions of the metric "APACHE:responseTime", for example, the region where the response time is "> 2000.0" would be assigned a value of 0, whereas the region where the response time is "≤ 2000.0" would be assigned a value of 100 (see Equation 3). This is because it is more desirable for the system to be in the region where the response time is not violated; i.e., "m_1.value ≤ 2000.0". Thus, given a measurement about a particular metric (i.e., m_i.value), Equation 3 assigns a numeric value corresponding to the appropriate metric's region where the measurement falls. It is the combination of these values over all the metrics that uniquely identify individual states.

Thus, if the policy of Figure 3 was the only policy in P, it would yield four states in our approach as illustrated in Table 2. In this case, state s_1 would be considered a "violation" state since it is the only situation which causes the policy to be violated, i.e., as a result of the violation of both policy conditions. Hence, actions set $A(s_1)$, in addition to action a_0 (i.e., do-nothing), would consist of the actions of the violated policy. The remaining three states are considered as "acceptable" states.

Example 2 *Suppose that, we extend Example 1 by adding the policy of Figure 8 (i.e., p_2) onto the policies set P such that $P = \{p_1, p_2\}$.*

It follows from Example 2 that the state metric m_1 ="APACHE:responseTime" would now be associated with two unique policy conditions; "APACHE:responseTime > 2000.0" from policy p_1 and "APACHE:responseTime < 250.0" from policy p_2. Consequently, the state metric "APACHE:responseTime" would now consist of three regions; i.e., "m_1.value < 250.0", "250.0 ≤ m_1.value ≤ 2000.0", and "m_1.value > 2000.0". In

m_i	c_k	Policy Condition	R_{m_i}	$R^j_{m_i}$	$f(R^j_{m_i})$
m_1	c_3	`APACHE:responseTime < 250.0`	$m_1.value < 250.0$	$R^1_{m_1}$	100
			$250.0 \leq m_1.value \leq 2000.0$	$R^2_{m_1}$	50
	c_1	`APACHE:responseTime > 2000.0`	$m_1.value > 2000.0$	$R^3_{m_1}$	0
m_2			$m_2.value \leq 0.0$	$R^1_{m_2}$	100
	c_2	`APACHE:responseTimeTREND > 0.0`	$m_2.value > 0.0$	$R^2_{m_2}$	0

Table 3. A metrics structure derived from the policy system of Example 2.

State	$R^k_{m_j}$		$f(R^k_{m_j})$		$A(s_i)$	
s_i	$R^k_{m_1}$	$R^k_{m_2}$	$f(R^k_{m_1})$	$f(R^k_{m_2})$	a_l	State action
s_1	$R^3_{m_1}$	$R^2_{m_2}$	0	0	a_0	γ-`action`
					a_1	`AdjustMaxClients(+25)`
					a_2	`AdjustMaxKeepAliveRequests(-30)`
					a_3	`AdjustMaxBandwidth(-128)`
s_2	$R^3_{m_1}$	$R^1_{m_2}$	0	100	a_0	γ-`action`
s_3	$R^2_{m_1}$	$R^2_{m_2}$	50	0	a_0	γ-`action`
s_4	$R^2_{m_1}$	$R^1_{m_2}$	50	100	a_0	γ-`action`
s_5	$R^1_{m_1}$	$R^2_{m_2}$	100	0	a_0	γ-`action`
					a_4	`AdjustMaxClients(-25)`
					a_5	`AdjustMaxKeepAliveRequests(+30)`
					a_6	`AdjustMaxBandwidth(+64)`
s_6	$R^1_{m_1}$	$R^1_{m_2}$	100	100	a_0	γ-`action`
					a_4	`AdjustMaxClients(-25)`
					a_5	`AdjustMaxKeepAliveRequests(+30)`
					a_6	`AdjustMaxBandwidth(+64)`

Table 4. Sample policy states based on the metrics structure of Table 3.

this case, the values assigned by Equation 3 to the above three regions would be 100, 50, and 0, respectively, as shown in Table 3. As a result, the policy system of Example 2 would yield six states in our approach as illustrated in Table 4 where states s_1, s_5, and s_6 would be considered as "violation" states whereas the remaining states would be considered as "acceptable" states. Thus, depending on the number of active policies in a set as well as the number of different metrics and different conditions on those metrics, the number of potential policy-states *could* be quite large; we comment further on this in Section 9. A key distinction between this and other related work (see, for example, [19, 20, 21, 22, 23]) is that the state structure is dependent only on the enabled expectation policies and can thus be automatically determined once a set of policies is specified.

5.2 System Transitions

Transitions are essentially determined by the actions taken by the management system and labelled by a value determined by our Reinforcement Learning algorithm. Which brings us to the next definition:

Definition 9 *Let $G^P = \langle S, T \rangle$ be a state transition graph for the policy system $PS = \langle P, W_C \rangle$ such that $t_i(s_p, a_p, s_c) \in T$. A state transition $t_i(s_p, a_p, s_c)$ is a directed edge corresponding to a transition originating from state s_p and ending on state s_c as a result of taking action a_p while in state s_p, and is labelled by $\langle \lambda, Q_{t_i}(s_p, a_p) \rangle$, where:*

- λ is the frequency (i.e., the number of times) through which the transition occurs.
- $Q_{t_i}(s_p, a_p)$ is the action-value estimate associated with taking action a_p in state s_p. In our current implementation, $Q_{t_i}(s_p, a_p)$ is computed using a one-step Q-Learning [3] algorithm (see Equation 4).

A change in the system's state may also be due to external factors other than the impact of the actions taken by the autonomic manager. In a dynamic Web server environment, for example, a transition may be a result of a request to a page with a database-intensive query, which could potentially cause a state transition. These are modeled in the state-transition graphs as γ-transitions; the actions responsible for such transitions are denoted by a_0 (i.e., γ-`action`) as illustrated in Table 4.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (4)$$

```
expectation policy{RESPONSETIMENormal(PDP, PEP)}
 if(APACHE:responseTime < 250.0)
 then{AdjustMaxClients(-25) test{newMaxClients > 49} |
      AdjustMaxKeepAliveRequests(+30) test{newMaxKeepAliveRequests < 91} |
      AdjustMaxBandwidth(+64) test{newMaxBandwidth < 1281}}
```

Figure 8. An expectation policy for dealing with an improvement in the server's response.

5.3 Reward Function

The main objective (or *goal*) of the autonomic manager, in essence, is to learn an optimal policy for "steering" the system towards "acceptable" states and away from "violation" states. In order to achieve this, a numeric reward, r, must be defined after each time step during which the agent acts (see, for example, Figure 1) to indicate the desirability of taking a particular action in a given situation (i.e., state s_t). What, then, should the reward be in order to encourage the learning of optimal behavior?

We are currently exploring one approach for deriving the reward signal, such that the learning agent is encouraged to take actions which may eventually lead to "acceptable" states. Rather than taking the simplest approach whereby an agent is only rewarded if an action results in a transition to such a state, we associate each state (both "violation" and "acceptable") with a reward value (derived from the state's metrics), which measures the desirability of the agent being in a particular state. We take this approach for three main reasons:

1. We do not make any assumptions about the accuracy of the enabled expectation policies since our main objective is to evaluate the effectiveness of these policies and, if necessary, adapt their use accordingly in order to meet specific objectives. We cannot assume, for example, that the use of an active set of expectation policies as is would be sufficient to effectively resolve the violations in QoS requirements; i.e., completely steer the system from "violation" to "acceptable" behavior.

2. The main objective of the learning agent is to figure out how to effectively use existing policies. Thus, while it may not always be possible to achieve the final objective based on the current set of active policies, the agent could still learn how at least to steer the system "towards" acceptable behavior. For example, if the objective (as defined by the enabled expectation policies) is to ensure that violations in the server's CPU and memory utilization are resolved, then we would consider a state where only a single metric is violated as better, i.e., closer to the acceptable behavior than, say, a state where both metrics are violated[1]. We could even go a step further by also considering the significance of state metrics. It could be that a violation in CPU utilization carries more weight than, say, that of memory utilization. Thus, the agent could be rewarded more generously for taking actions which result in no violation in CPU utilization, but less generously if those actions lead to no violations in memory utilization. We elaborate further on this in the next section.

3. The dynamicity of the system in terms of the changes in the state structure as a result of run-time policy modifications necessitates more flexibility in terms of how the reward function is derived. This is the focus of our current research on adaptation strategies and is beyond the scope of this paper.

Thus, we associate each state with a reward whose value increases towards acceptable behavior. From the example above, a reward is zero if the action leads to a state where both CPU and memory utilization are violated (since $f(R^j_{m_i})$ is 0 for both metrics), and is the highest for a state with no violation (since $f(R^j_{m_i})$ is 100 for both metrics). And this brings us to our next definition.

Definition 10 *Given the current system state $s_t = \langle \mu, M(s_t), A(s_t) \rangle$, such that $m_i \in M(s_t)$; an agent visiting state s_t after taking action a in the previous state is rewarded as follows:*

$$r(s_t) = \sqrt{\sum_{i=1}^{n} m_i.\omega \times [f(R^j_{m_i})]^2} \qquad (5)$$

where, n is the number of metrics, and $m_i.\omega$ and $R^j_{m_i}$ correspond, respectively, to the weight associated with metric m_i and the region where metric m_i measurement falls (see Definition 8). In essence, Equation 5 assigns each state a reward whose value increases as one moves towards the most desirable states.

[1] In this example, there would be four states since each state metric could have two possible regions; either it is violated or not. Thus, the following states are possible; (i) a state where both metrics are violated, (ii) a state where only CPU utilization is violated, (iii) a state where only memory utilization is violated, and (iv) a state where neither metric is violated.

6 Learning by Reinforcement

Central to the functionality of the PDP is the need to determine what actions to take given certain violations in QoS requirements. Note that the choice of actions, $a \in A(s)$, is dependent on the expectation policies that are violated when the system is in state s. Since policy violations are triggered by Monitor events collected during the current management interval, the PDP must decide whether to base its action selection decisions on this information alone or whether to request advice based on past experience when making those decisions. This decision-making dilemma lends itself well to the *explore-exploit* dilemma in Reinforcement Learning and is explored in detail next. But first, we comment briefly on key characteristics that could influence the choice of the algorithm for balancing exploration and exploitation.

- Each action $a \in A(s)$ cannot be treated equally, particularly because of the importance of the order in which the actions are specified within each expectation policy. As illustrated by the the expectation policy of Figure 3, it is often the case that more drastic actions (i.e., `AdjustMaxBandwidth(-128)` which throttles clients requests by reducing server's network bandwidth) are taken once it is no longer possible, for example, to meet the specified objectives through the adjustment of applications tuning parameters. It is therefore important that, to some extent, this order is preserved, at least during the initial phase of the learning process.

- It is often the case that characteristics specific to the violations (and not just the type of violation) provide useful information about the state of the system as well as how to best respond to the situation. For example, if the aim is to ensure that the server's response does not exceed 2000.0 ms, then it might be desirable to treat a violation in the server's response time of 5000 ms differently than, say, a violation of 2001 ms. Such kind of information could also be useful in guiding the exploration process to ensure that more urgent needs are addressed first.

- We note also that exploration could be quite costly especially in situations where excessive penalties are incurred. It is, therefore, important that the exploration process takes advantage of existing knowledge about the policies and violation events as opposed to selecting policy actions based exclusively on the type of violations.

To this end, we propose the use of a *near greedy* approach to balancing exploration and exploitation whereby the learning agent behaves greedily - by executing the action with the highest $Q(s, a)$ - most of the time (with probability $1 - \epsilon$) and, once in a while (with probability ϵ) the agent selects an action independent of the current action-value estimate $Q(s, a)$. Unlike the ϵ-*greedy* method [3] which treats all actions equally during exploration, action selection is based on the action-value estimate that is derived from the characteristics of both policies and violations. This is particularly useful when it is necessary to differentiate one action from another given that multiple, and at times conflicting, actions may be "advocated" by the violated policies and where the order in which the actions are specified might be of importance.

6.1 Exploration Strategy

In certain situations, the learning agent may need to make management decisions without depending, exclusively, on past experience. This could be part of the agent's strategy of exploring its environment to discover what actions bring the most reward. It could also be because the agent may have no other choice if past experience does not include knowledge about the current situation if, in fact, it is the first time the situation is encountered. Consequently, the agent may have to base its decisions on information other than past experience. In our approach, these decisions are guided by the following strategies that are based on the characteristics of the enabled expectation policies and those of the violation events:

1. *The severity of the violation:* Rather than treating each violation equally, we assign more weight to those violations that are more severe. The severity of the violation is based on the value of the metric relative to the condition's threshold. For example, for a CPU utilization of 100% given the condition "`CPU:utilization > 85.0`" (i.e., as a result of violating the policy of Figure 9), this value is computed from the difference between the measured value and its threshold value (i.e., 15%) as defined by Equation 8.

2. *The significance of the violation:* In the case that multiple policies are violated, it may be desirable to assign a higher priority (or weight) to a particular event so that the management system can respond to such a violation (i.e., by selecting appropriate policy actions) first before dealing with other less-important violations. For instance, it is quite reasonable to respond to CPU utilization violations before addressing violations related to, say, response time since failure to address the former may result in more severe violations of the latter as a result of over-utilization of CPU resources. This is done by allowing a weight to be associated with events which then become weights on the conditions that become true in violated policies (see Definition 2). The weight associated with policy con-

```
expectation policy{CPUViolation(PDP, PEP)}
 if(CPU:utilization > 85.0) & (CPU:utilizationTREND > 0.0)
 then{AdjustMaxClients(-25) test{newMaxClients > 49} |
      AdjustMaxKeepAliveRequests(-30) test{newMaxKeepAliveRequests > 1} |
      AdjustMaxBandwidth(-128) test{newMaxBandwidth > 255}}
```

Figure 9. An expectation policy for resolving Apache's CPU utilization violation.

```
expectation policy{CPUandRESPONSETIMEViolation(PDP, PEP)}
 if(CPU:utilization > 85.0) & (CPU:utilizationTREND > 0.0) &
   (APACHE:responseTime > 2000.0) & (APACHE:responseTimeTREND > 0.0)
 then{AdjustMaxKeepAliveRequests(-30) test{newMaxKeepAliveRequests > 1} |
      AdjustMaxBandwidth(-128) test{newMaxBandwidth > 255}}
```

Figure 10. An expectation policy for resolving Apache's CPU utilization and response time violations.

dition c_i which then becomes the strength of policy p_j is denoted by the parameter $c_i.\omega$ (see Equation 7).

3. *The advocacy of the action:* In the case that multiple policies are violated, it might be possible that more than one policy advocates the same action. For example, in our current test environment involving the Apache server and other components, different policies with different conditions (see, for example, Figures 3 and 9) may indicate that the same action be taken, i.e., `AdjustMaxBandwidth` which controls the maximum number of requests a server can process. The number of policies advocating the action as well as the position of the action within each policy (whose weight is denoted by the parameter $W_a(p_j)$ in Equation 6) are also considered when estimating $Q_0(s, a)$. The position is of particular interest since, in our experience, it is often the case that more drastic actions are not taken until other actions to adjust tuning parameters have first been "tried".

4. *The specificity of the policy:* In a situation where several policies are violated, the number of conditions within each policy (as well as conditions weights) could also be taken into consideration when determining which policy has more weight. For example, in the event that both CPU utilization and response time are violated, the policy in Figure 10 would be given more weight than the policy of Figure 9. This information could be taken into account when evaluating the strength of policy p_j, which we refer to as $S(p_j)$ (see Equation 7).

Thus, given the policy system $PS = \langle P, W_C \rangle$ (see Definition 2) and supposing that P_v is a set of expectation policies that are violated in the current management interval such that $P_v \subseteq P$, we can estimate the initial value of an action, a, as follows:

$$Q_0(s,a) = \frac{\sum_{p_j \in [P_v]_a} tanh[S(p_j)] \times W_a(p_j)}{\| [P_v]_a \|} \quad (6)$$

where $[P_v]_a$ is the subset of violated policies advocating action a; $W_a(p_j)$ is the weight of action a based on its position within policy p_j. In our current implementation, actions weights take values between 100 and 0 such that the first policy action gets the highest value (i.e., 100) while the last policy action gets the lowest value (i.e., 0), with weights assigned to the actions at equal intervals according to Equation 3. Thus, in the case of a policy with three actions such as the policy of Figure 9, the values would be 100, 50, and 0, in that order; $S(p_j)$ is the strength of policy p_j as specified by Equation 7:

$$S(p_j) = \sum_{c_i \in p_j} c_i.\omega \times V(c_i) \quad (7)$$

where $c_i.\omega$ is the weight associated with policy condition c_i based on the significance of the condition's violation (see Definition 2), and $V(c_i)$ is the severity of the violation of condition c_i. This value is computed as follows:

$$V(c_i) = \left| \frac{e_i.\texttt{value} - c_i.\Gamma}{\Omega} \right| \quad (8)$$

where $e_i.\texttt{value}$ is the current value of the event responsible for violating condition c_i, $c_i.\Gamma$ is the threshold value of condition c_i, and

$$\Omega = \begin{cases} 1, & |c_i.\Gamma| \leq 1 \\ c_i.\Gamma, & otherwise \end{cases} \quad (9)$$

Briefly, Equation 7 estimates $Q_0(s, a)$ based on the *severity* of the violations of the conditions associated with, as well as the *significance* of, individual policies. We measure the severity based on the difference between the condition's threshold, $c_i.\Gamma$, and the observed value of the metrics, $e_i.\texttt{value}$, responsible for its violation (see Equation 8). In certain situations, it may be desirable to designate a higher priority to a particular event so that the management system can respond to such a violation first (i.e., by selecting appropriate policy actions) before dealing with other less important violations. This is the purpose of the parameter $c_i.\omega$ in Equation 7. This information could then be used to estimate the action value (see Equation 6), which takes into account the number of policies advocating the action, the position of the action, and the severity associated with the violation of the conditions within each violated policy. Thus, the same set of violations, for example, may result in different actions being taken depending on the initial action-value estimates. This is in contrast to static approaches where the order of the actions is always the same for the same set of violations.

6.2 Exploitation Strategy

As noted previously, it is often very difficult to obtain, in advance, models that accurately capture systems dynamics particularly for the state of the enterprise systems. Our approach to learning, for reasons mentioned in Section 2, is based upon the Dyna-Q framework [8] where the model of the system is continuously learned, on-line, and used for planning. We are currently exploring several strategies on how such a model, represented by the state-transition graph (as discussed in Section 5), might be used to help the system adapt the way it uses policies when making decisions on how to resolve QoS requirements violations. These strategies fall into two broad categories:

1. **Reactive Enforcement**: In this approach, the autonomic manager could adapt the way it reacts to violations in QoS requirements (i.e., respond after a violation has occurred) based on the currently learned model. One such approach involves having the PDP request advice from the learning component during each management cycle where the system is in "violation" state. This may include, for example, an advice on what policy action to take in the current state (i.e., s) based on the currently learned $Q(s, a)$ estimates associated with each action $a \in A(s)$. It may also be possible to recommend multiple actions if their impact is deemed positive. This may involve, for example, computing the shortest path from the current "violation" state to an "acceptable" state based on the $Q(s, a)$ estimates associated with the actions within the current state-transition graph. A path, in this case, constitutes an ordered list of actions. For instance, if the system is in state s_1 of Figure 7, the learning agent may recommend the enforcement of a set of actions consisting of $\{a_3, a_2, a_2\}$ essentially steering the system to an "acceptable" state s_8.

2. **Proactive Enforcement**: In this approach, the autonomic manager, in anticipating possible violations in QoS requirements, may recommend a set of actions aimed at steering the system away from "violation" states before the system gets there. For instance, if it has been observed that the system makes a γ-transition from an "acceptable" state (i.e., s_2 in Figure 7) to a "violation" state (i.e., s_1) with a very high probability, then appropriate actions could be taken before the system gets to state s_1. Thus, actions $\{a_3, a_2, a_2\}$ could be enforced while the system is still in state s_2 which may, as a result, move the system to a more stable "acceptable" state (i.e., s_8) consequently minimizing possible future violations.

The above two approaches to QoS provisioning highlight several key advantages on how the autonomic management system can respond to violations: First, rather than restricting the selection of policy actions to only those advocated by the violated policies (i.e., $a \in A(s)$), the autonomic manager is able to look beyond the actions within a single state for actions, some of which might not even be part of those in the violated policies, whose impact may be positive but not immediate. Second, the autonomic manager could take multiple actions. Assume, for example, that the system is in state s_1 and that a_2 corresponds to the action "`AdjustMaxClients(+25)`" as specified by the policy of Figure 3. Thus, instead of increasing `MaxClients` by 25, the same action could be performed twice. A key advantage here is that multiple adjustments to the tuning parameters could be made when past behavior suggests that it is likely prudent to do so. Third, the autonomic manager has the ability to be proactive, that is, use past experience to take actions in anticipation of policy violations. This would be done by looking ahead in the state graph. The agent may determine whether some action could lead to either a very bad situation or a very good one. For instance, the agent using the state-transition information in Figure 7 could avoid actions such as a_2 while in state s_1 if past experience show that, once that action is taken, it is less likely for the system to make a transition back to an acceptable state.

6.3 The Learning Algorithm

To compute the action-value estimates, we use a modified version of the Dyna-Q algorithm (see Algorithm 1) that enables the agent to learn in non-deterministic environments. The algorithm (see Algorithm 2), which we refer to

herein as Dyna-Q*, takes into account transition probabilities when computing action-value estimates.

Algorithm 2 Dyna-Q* Algorithm

Input: Initialize $G^P = \langle S, T \rangle$ for policy system $PS = \langle P, W_C \rangle$

for $i = 1$ *to* ∞ **do**
  $s \leftarrow$ current (non terminal) state
  $a \leftarrow \epsilon$-greedy(s, Q_0) (see Equation 6)
  Execute a; observe resultant state, s'
  $Q(s,a) \leftarrow Q(s,a) + \alpha\{E[r(s,a)] + \gamma E[\max_{a'} Q(s',a')] - Q(s,a)\}$
  $G^P \leftarrow \langle s', t(s,a,s') \rangle$
  for $j = 1$ *to* k **do**
    $s \leftarrow$ random previously observed state
    $a \leftarrow$ random action previously taken in s
    $Q(s,a) \leftarrow Q(s,a) + \alpha\{E[r(s,a)] + \gamma E[\max_{a'} Q(s',a')] - Q(s,a)\}$
  end for
end for

To compute the expected values, we define a transition probability as follows:

$$Pr[t(s,a,s')] = \frac{t_i(s,a,s').\lambda}{\sum_{t_i(s,a,s'_j) \in T(s)} t_i(s,a,s'_j).\lambda} \quad (10)$$

where $t_i(s,a,s').\lambda$ is the frequency associated with a transition originating from state s and terminating at state s' as a result of taking action a in state s (see Definition 9). Thus, $t_i(s,a,s'_j) \in T(s)$ is a subset of transitions originating from s (i.e., $T(s)$) as a result of taking action a. From Equation 10, the expected reward can be computed as follows:

$$E[r(s,a)] = \sum_{t_i(s,a,s'_j) \in T(s)} Pr[t_i(s,a,s'_j)] \times r(s'_j) \quad (11)$$

where $r(s'_j)$ is the reward associated with state s'_j computed using Equation 5. Similarly, the expected action-value estimate can be computed as follows:

$$E[\max_{a'} Q(s',a')] = \sum_{t_i(s,a,s'_j) \in T(s)} Pr[t_i(s,a,s'_j)] \times \max_{a'} Q(s'_j,a') \quad (12)$$

Note that, in the case of deterministic transitions, $Pr[t_i(s,a,s')] = 1$. Thus, $E[r(s,a)]$ is essentially equal to $r(s'_j)$; i.e., the reward the agent receives after making a transition to state s'_j (see Definition 10). Similarly, $Q(s,a)$ is essentially the same as the action-value estimate associated with the transition (i.e., $Q_{t_i}(s,a)$) as computed by Equation 4. And this is consistent with the implementation of the Dyna-Q Algorithm in deterministic environments as described in Section 2 (see Algorithm 1).

7 Results and Experience

This section presents the prototype implementation of the adaptive policy-driven autonomic system as well as report on our experience.

7.1 Managed System

We evaluated the effectiveness of the learning mechanisms on the behavior of a multi-component Web server consisting of an Apache (v2.2.0) [24] which was configured with a PHP (v5.1.4) module [25], and a MySQL (v5.0) database server [26]. We used the PHP Bulletin Board (phpBB) application [27] to generate dynamic Web pages. This application utilizes queries to display information stored inside a database, in our case, the MySQL database. The main database tables include forums, topics, posts, users, and groups. These tables are used to store information specific to discussions. In addition to viewing forum-related information, users may post messages using forms, which can be viewed through a Web browser. A single workstation was used to host the components as illustrated in Figure 12. Service differentiation mechanisms for classifying gold, silver, and bronze clients were also implemented (see Section 7.3). Several effectors were implemented and included those for adjusting the following parameters: (For a detailed description of the tuning parameters excluding `MaxBandwidth`, the reader is referred to [24, 26].)

- `MaxClients` (Apache): controls the maximum number of server processes that may exist at any one time (i.e., the size of the worker pool) and corresponds to the number of simultaneous connections that can be serviced. Setting this value too low may result in new connections being denied. Setting it too high, on the other hand, allows multiple clients' requests to be processed, but may lead to performance degradation as a result of excessive resource utilization.

- `MaxKeepAliveRequests` (Apache): corresponds to the maximum number of requests that a *keep-alive* connection [28] can transmit before it is closed. Its value is often set relative to the `KeepAliveTimeout`, which corresponds to the client's think time - the amount of time, in seconds, the server will wait on a persistent connection before closing it. Setting this value too high may result in having connections linger for too long after a client has disconnected thus wasting server's resources. On the other hand, setting this value too low may lead to having clients rebuild their connections often, possibly impacting the response time and CPU utilization.

- `EaccMemSize` (PHP): The PHP performance was further enhanced with the eAccelerator [29] encoder.

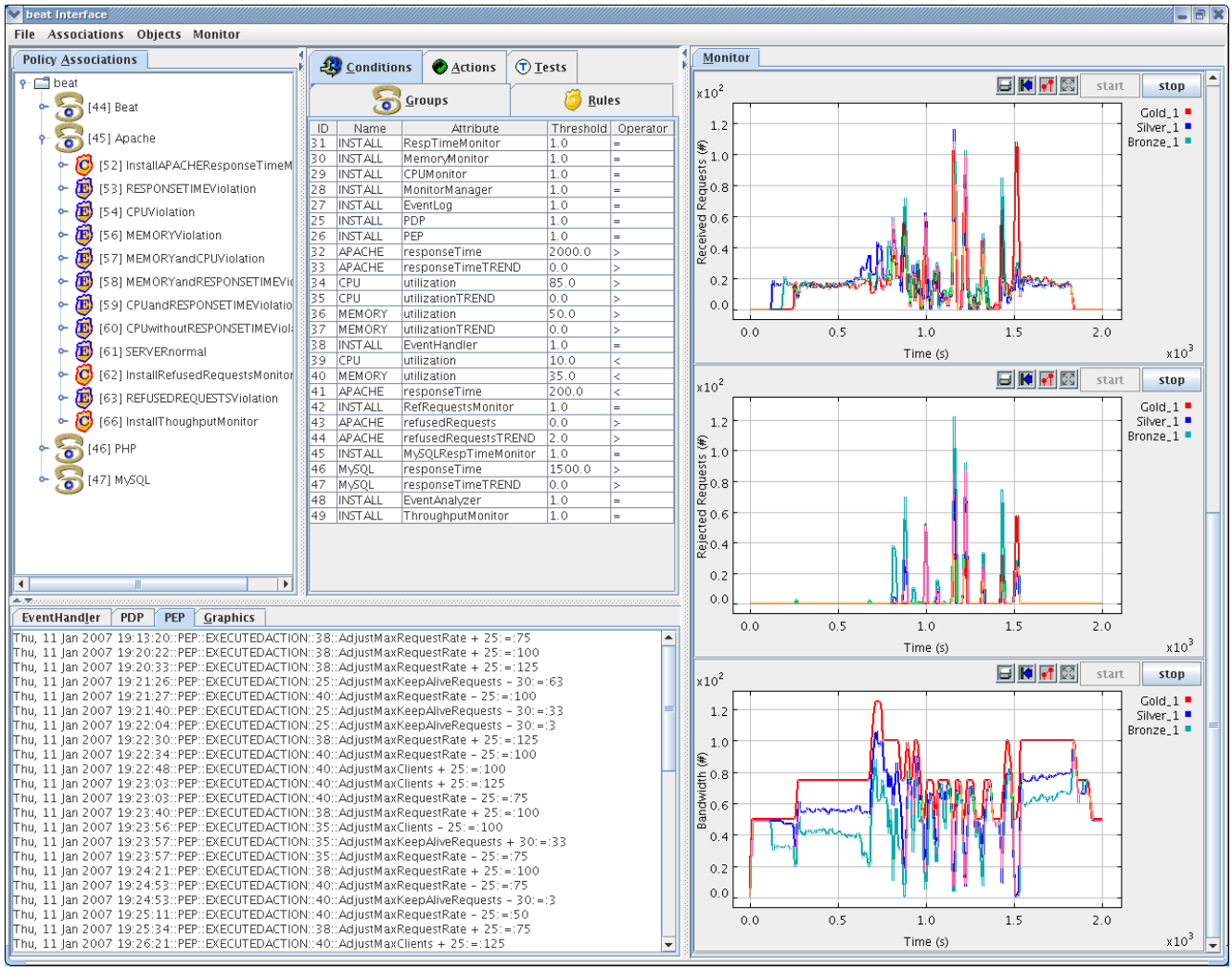


Figure 11. A Graphical User Interface (GUI) to the autonomous management system.

This module provides mechanisms for caching compiled scripts so that later requests invoking similar scripts do not incur compilation penalty. The parameter `EaccMemSize` controls the size of memory cache.

- `KeyBufferSize` (MySQL): corresponds to the total amount of physical memory used to index database tables.
- `ThreadCacheSize` (MySQL): corresponds to the number of threads the database server may cache for reuse. Thus, instead of creating a new thread for each request to the database, the server uses the available threads in the cache to satisfy the request. This has the advantage of improving the response time as well as the CPU utilization.
- `QueryCacheSize` (MySQL): corresponds to the maximum amount of physical memory used to cache query results. Thus, a similar query to previously cached results will be serviced from memory and not from disk.
- `MaxConnections` (MySQL): corresponds to the maximum number of simultaneous connections to the database.
- `MaxBandwidth` (System): corresponds to the physical capacity (in kbps) of the network connection to the workstation hosting the servers.

The servers provide support for dynamic adjustment of the parameters. For the Apache-PHP server, for example, the actual adjustment to the parameters was done by editing the appropriate configuration file and performing a *graceful restart* [24] of the server.

7.2 Using Policies

We used the Policy Tool of Figure 11 to specify policies which expressed the desired behavior of the managed system (in terms of CPU, memory utilization, and response time thresholds) as well as possible management actions to be taken whenever those objectives were violated (see, for example, the policy of Figure 3). We also defined several policies that dealt specifically with the optimization of resource usage whenever an opportunity arose. This is illustrated by the policy of Figure 8 where, given that there are no violations in QoS requirements, one might reduce the number of `MaxClients` to a smaller value, thus reducing memory utilization. During this time, existing clients might also be allowed to hold onto server processes for much longer (i.e., by increasing `MaxKeepAliveRequests`) to improve their response time (rather than requiring them to re-negotiate their connections every so often). One might also increase the server's bandwidth. In our implementation, we classify states associated with the violations of such policies as "acceptable" (see Definition 7). Each state consisted of ten metrics corresponding to equally weighted (see Definition 2) conditions from the enabled policies.

7.3 Testbed Environment

A testbed environment consisted of a collection of networked workstations, each connected via 10/100 megabit-per-second (Mbps) Ethernet switch (see Figure 12). They include an administrative console used to run the Policy Tool; a Linux workstation with a 2.0 GHz processor and 2.0 Gigabytes of memory which hosted the Apache Web Server along with the Knowledge Base and the MySQL database server; and three workstations used to run the traffic load tool for generating server requests for the gold, silver and bronze service classes.

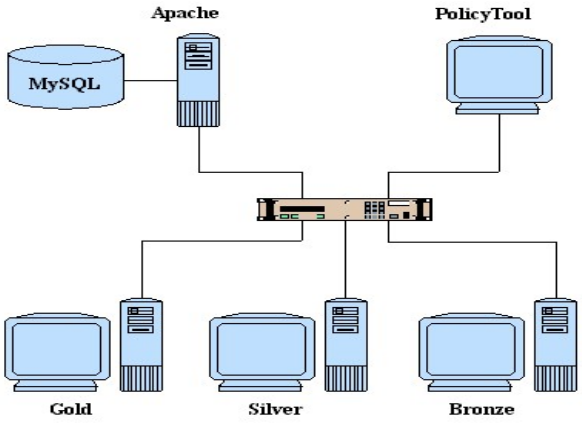


Figure 12. Testbed Environment.

In order to support service differentiation, a Linux Traffic Controller (TC) Tool [30] was used to configure the bandwidth associated with the gold, silver, and bronze service classes. Thus, given the maximum possible bandwidth the service classes throughput were assigned proportionately according to the ratio 85:10:5; bandwidth sharing was also permitted. The actual classification was based on the remote IP address of the clients' request and occurred at the point where requests reached the workstation hosting the Apache server. The tuning parameter `MaxBandwidth` is what determines how much bandwidth is assigned to each service class. Thus, given that the policy of Figure 3 has been violated and that it is no longer possible, for example, to adjust the parameters `MaxClients` and `MaxKeepAliveRequests`, then the last policy action (i.e., `AdjustMaxBandwidth(-128)`) would be executed, which essentially reduces the total bandwidth by 128 kbps. The percentage of the new bandwidth is what is eventually assigned to the different service classes.

7.4 Workload Generator

To simulate the stochastic behavior of users, the Apache load generator tool (ab) [24] was modified to support concurrent and independent keep-alive requests to the server. The tool was also modified to emulate the actual behavior of users by traversing the Web graph of an actual Web site. Thus, for each response from the server, the tool randomly selects which subsequent link (among the links in the received Web page) to follow. In the experiments reported in this paper, we only considered requests involving dynamic Web content through the use of the phpBB application. Also, we only considered database *read-only* requests. For all the experiments, the load generator in each client's workstation was configured such that the number of concurrent connections to the server and the think-time for the gold, silver, and bronze clients were identical. These values were set to ensure that the server was under overload conditions (i.e., saturated) for the duration of the experiment.

7.5 Experiments and Results

To evaluate the impact of the learning mechanisms on the behavior of the server - which was measured in terms of Apache's responsiveness (i.e., response time), throughput (i.e., number of requests processed), and resources utilization (i.e., CPU and memory) - we conducted three experiments: The first (base) experiment (*Exp-1*) looked at the behavior when all the expectation policies were disabled. The server's bandwidth was also set arbitrarily large and service differentiation mechanisms were disabled. The second experiment (*Exp-2*) looked at the impact of the action selection mechanisms (see Equation 6) which depended ex-

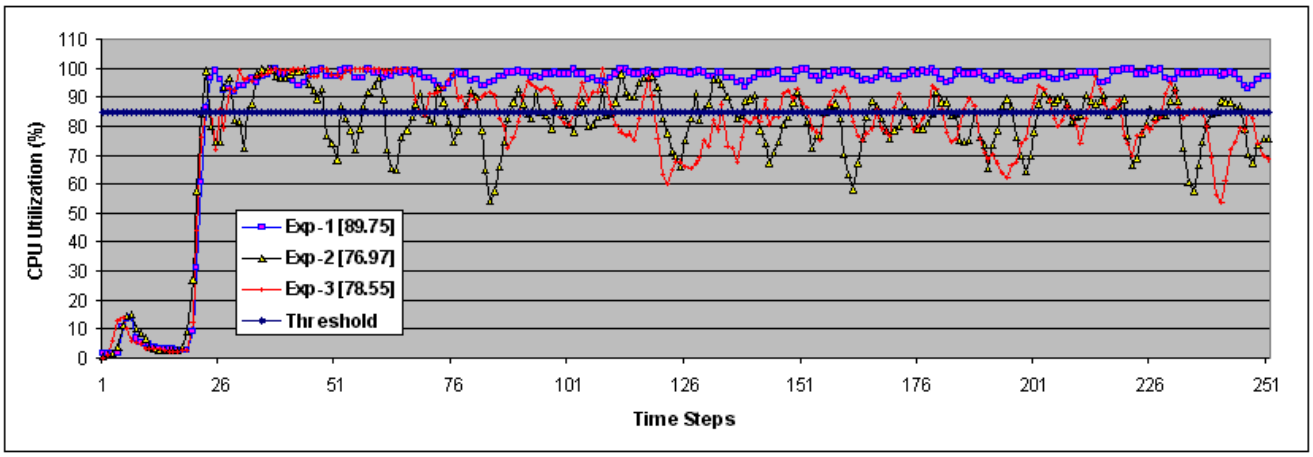


Figure 13. Server's CPU utilization measurements.

clusively on the characteristics of both the violation events and the violated policies within a single management interval; i.e., without the learning mechanisms. The third experiment (*Exp-3*) looked at the impact of action selection mechanisms based on learning from past experience in the use of policies. We used the areas occupied by the curves beyond the thresholds (85% for CPU utilization, 50% for memory utilization, and 2000 ms for response time) to compare the performance of the server relative to the base experiment (i.e., Exp-1). This provided a measure of the amount of time the system spent in "violation" states, essentially allowing us to compare performance improvement relative to the base experiment.

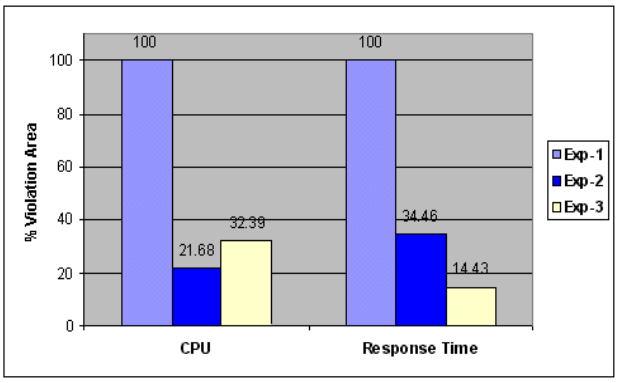


Figure 14. Area beyond the thresholds.

7.5.1 CPU Utilization

Figure 13 compares the behavior of the server in terms of CPU utilization. The number listed in square brackets beside each experiment is the average utilization for the duration of the experiment. From these results, we can see that the average CPU utilization for the base experiment (i.e., Exp-1) fell above the threshold value (i.e., 85%) whereas that of Exp-2 and Exp-3 fell below the threshold. While the main objective was to ensure that CPU utilization did not exceed 85% (which was accomplished in both Exp-2 and Exp-3), it is worth noting that action-selection mechanisms based on learning from past experience in the use of policies (i.e., Exp-3) performed slightly worse than when no learning mechanisms were enabled (i.e., Exp-2). This became more obvious when we considered the area occupied by the graphs above the thresholds relative to the base experiment as illustrated in Figure 14.

There are several reasons for this: The most obvious is probably the impact of γ-`action` (see, for example, Table 2) particularly during the initial stages of the learning process whereby the agent may be forced to spend more time *exploring* its environment (while building up the model). This may include trying actions such as γ-`action`; i.e., doing-nothing instead of performing actual adjustments to the tuning parameters to resolve QoS violations. This stage can clearly be seen from the graph of Exp-3 in Figure 13; i.e., between time-steps 26 and 70. The less obvious reason relates to the fact that the agent may have to consider multiple, and at times competing, objectives and this might be the best way of optimally meeting all the objectives. Thus, while the server may have performed slightly worse in Exp-3 than in Exp-2, the reverse was also true when considering the server's response time (see Figure 14) and throughput (see Figure 18). This is an illustration of one of the key challenges facing autonomic systems; i.e., how to negotiate between seemingly conflicting objectives: On the one hand, striving to meet customer

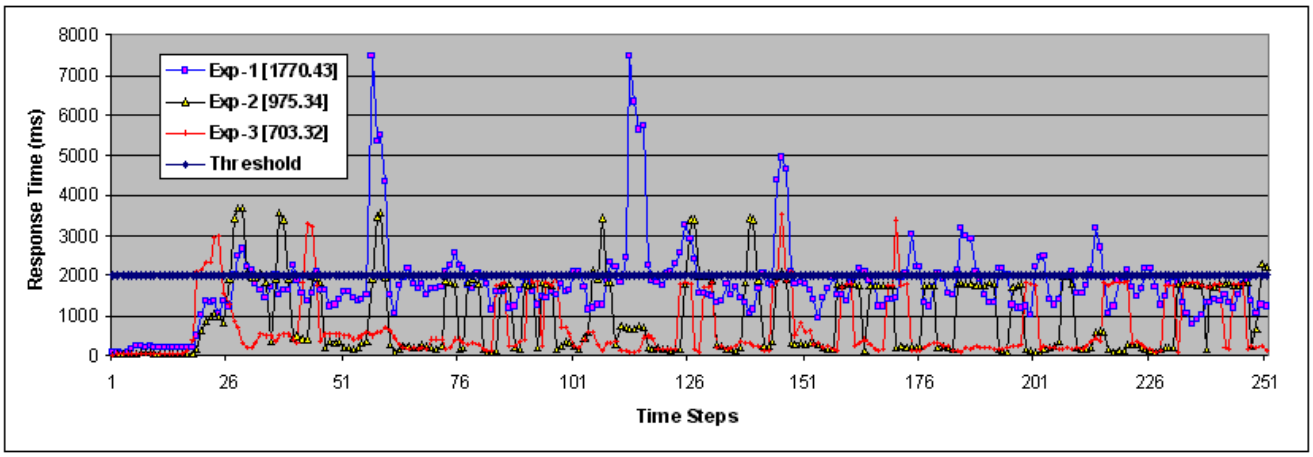


Figure 15. Server's response time measurements.

needs, in this case improving server's response time; on the other hand, trying to ensure efficient operation of systems and utilization of services.

7.5.2 Response Time

Response time measurements on the server side corresponded to the amount of time requests from non *keep-alive* connections spent on the waiting queue before they were served. The results are depicted in Figure 15 which also shows the average response for each experiment listed inside square brackets. From these measurements, one can see a significant improvement in the server's response time. This became more obvious when we computed the area above the thresholds relative to the base experiment as illustrated in Figure 14 where Exp-2 recorded at least a 65% improvement while Exp-3 recorded at least an 85% improvement in response time.

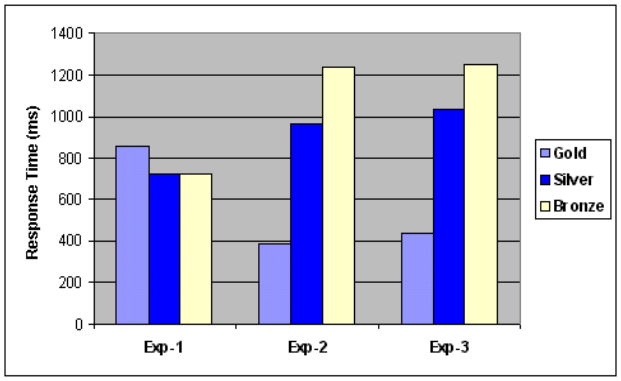


Figure 16. Client's response time.

We also compared client-side response time measurements which calculated the average time it took for a client to receive a response from the server (see Figure 16). Since no service differentiation mechanisms were enabled for the base experiment (i.e., Exp-1), the measured response was somewhat similar for gold, silver, and bronze clients. However, this changed significantly in Exp-2 and Exp-3 where gold clients response time was significantly better than that of silver and bronze clients. We also observed significant improvement in the response time of gold clients in Exp-2 and Exp-3 compared to the average of Exp-1. However, between the two experiments, there was very little difference when similar service classes were compared.

7.5.3 Throughput

Throughput measurements looked at the average number of requests serviced by the server for the duration of the experiment. The results specific to Exp-3 are shown in Figure 17: results for all the three experiments are summarized in Figure 18. Again, since no service differentiation mechanisms were enabled in Exp-1, the measurements were essentially similar for the three service classes. Furthermore, comparing the average across service classes (see the values listed inside square brackets in Figure 18), one can see that slightly more requests were serviced in Exp-1 than in Exp-2 and Exp-3. This was expected since there weren't any restrictions, for example, in terms of the server's resource utilization. In terms of the performance of individual service classes for both Exp-2 and Exp-3, the throughput measurements were consistently higher for the gold than for the silver and bronze service classes. The server also performed consistently better in Exp-3 than in Exp-2 across service classes.

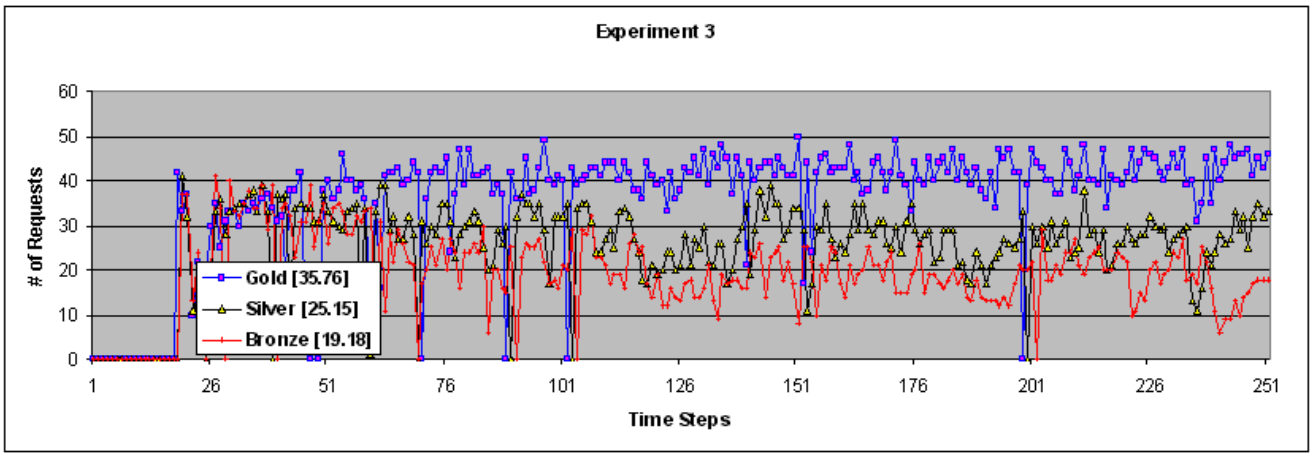


Figure 17. Server's throughput measurements.

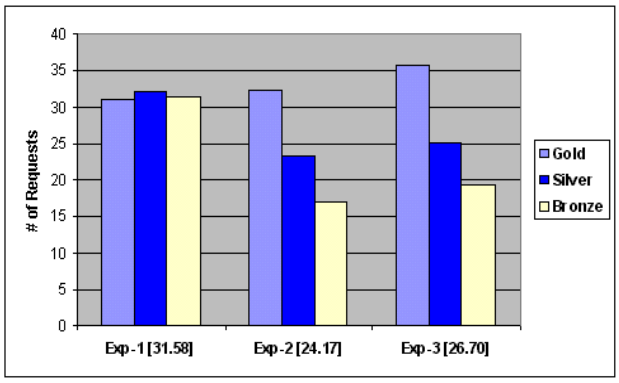


Figure 18. Server's throughput.

8 Related Work

Recently, several approaches based on Reinforcement Learning have been proposed for managing systems performance in dynamic environments. This section reviews some of the research work in this area and contrast them to our approach.

The work in [21] proposes the use of Reinforcement Learning for guiding server allocation decisions in a multi-application Data Center environment. By observing the application's state, number of servers allocated to the application, and the reward specified by the SLA, a learning agent is then used to approximate $Q_\pi(s, a)$. To address poor scalability in large state spaces, the authors initially proposed an approximation of the application's state by discretizing the mean arrival rate of page requests [20]. In their most recent work [21], they address this shortfall by proposing an off-line training, to learn function approximators using SARSA(0) [3], based on the data collected as a consequence of using a queuing-model policy, π, on-line. A key assumption is that the model-based policy is good enough to give an acceptable level of performance.

The authors in [22] propose a framework which make use of Reinforcement Learning methodologies to perform adaptive reconfiguration of a distributed system based on tuning the coefficients of fuzzy rules. The focus is on the problem of dynamic resource allocation among multiple entities sharing a common set of resources. The paper demonstrates how utility functions for making dynamic resource allocation decisions, in stochastic dynamic environments with large state spaces, could be learned. The aim is to maximize the average utility per time step of the computing facility through the reassignment of resources (i.e., CPUs, memory, bandwidth, etc.) shared among several projects.

The work in [23] proposes the use of Reinforcement Learning techniques in Middlewares to improve and adapt the QoS management policy. In particular, a Dynamic Control of Behavior based on Learning (DCBL) Middleware is used to learn a policy that best fits the execution context. This is based on the estimation of the benefit of taking an action given a particular state, where the action, in this case, is a selection of a QoS level. It is assumed that, each managed application offer several operating modes from which to select, depending on the availability of resources.

Our approach differs in several ways; First, the model of the environment is "learned" on-line and used, at each time-step, to improve the policy guiding the agent's interaction with the environment. Second, our strategy for adapting the use of policies makes use of a learning signal that is based only on the structure of the policies and should, thus, be applicable in other domains. Similarly, changing policies dynamically means that the heuristics will still work

for a new set of policies. Since the state signal is dependent only on the enabled expectation policies, its structure and size can also be automatically determined once a set of policies is specified. Third, we do not make use of policies which are by themselves "models" of the system being managed. While steady-state queuing models have received significant interest in on-line performance management and resource allocation in dynamic environments, we note that most of these approaches model the behavior of the application using the mean requests arrival rate, ignoring other important characteristics. In dynamic Web environments, for example, requests to dynamic pages with database intensive queries could stress the application significantly different (in terms of server's response, resources utilization, etc.) compared to, say, requests to static pages under the same rate. Our policies, on the other hand, are simpler and do not make any assumptions about workload characteristics. Fourth, our approach does not make any assumption about the accuracy of the policies used to drive autonomic management. We view learning as an incremental process in which current decisions have delayed consequences on how the learning agent behaves in future time-steps. It is significantly important, therefore, for training to be performed on-line in order for the agent to learn from the consequences of its own decisions and, if necessary, dynamically adapt the policy guiding its interaction with the environment.

9 Conclusion

In this paper, we have proposed a strategy for determining how to best use a set of active policies to meet the different performance objectives. Our focus has particularly been on the use of Reinforcement Learning methodologies to determine how to best use a set of policies to guide autonomic management decisions. Such use of learning has significant ramifications for policy-driven autonomic systems. In particular, it means that system administrators no longer need to manually embed system's dynamics into policies that drive autonomic management. Unlike previous work on the use of action policies, for example, which required system administrators to manually specify policy priorities for resolving run-time policy conflicts, desirable behavior could be learned. It should be noted, however, that, while Reinforcement Learning offers significant potential benefits from an autonomic computing perspective, several challenges remain when these approaches are employing in real-world autonomic systems. This section looks at how we intend to address some of these challenges.

9.1 Challenges

The choice of how to model system states has significant impact on the learning process. As with many real-world systems, the state space can become prohibitively large since its size increases exponentially with the number of state metrics and their discretization. As such, storing and analyzing statistics associated with each state may require significant computation resources, which could be exceedingly costly to implement in a live system. In our current approach, we make an approximation in the representation of the system's state by mapping the conditions of the enabled expectation policies onto the state metrics. A system where each state has ten metrics, each with two possible regions (i.e., "violation" and "acceptable"), for example, would have 2^{10} (1024) possible states. We note that, in such a system, a majority of the states are likely to correspond to "acceptable" system's behavior. This is illustrated in Table 2 where out of the four states, only one state (s_1) is considered a "violation" state since it is the only state that results in the violation of the policy in Figure 3. Thus, while the size of the state space could be large, many of these states may be considered as "goal" states and, as such, would have no actions associated with them. Furthermore, it is not guaranteed that the agent would visit all the possible states during the learning process. An immediate consequence of this is a reduction in the amount of information associated with states and their transitions.

As was noted previously, the change in the system's state might be a result of external factors other than the consequences of the actions of the agent. In a Web-server environment, such transitions are often triggered by changes in workload characteristics. For example, a sudden increase in the number of clients could trigger a transition to a violation state. The fact that the state signal is not derived from such characteristics means that the learning agent can not be certain about whether or not the system's behavior at time $t+1$ is the consequence of its action at time t. We have taken the approach of excluding requests characteristics from the state signal mainly due to the stochastic nature of the interactions between these characteristics and the behavior of the system. For instance, the number of concurrent connections, the type of request (i.e., static vs dynamic), the requests rate, etc., all these could have significant ramifications on the behavior of the server. Including these characteristics as part of the state signal is likely to add significant overhead in the learning process. Excluding such characteristics, on the other hand, will not hinder the learning process since in the long run, the agent would learn about the impact of the action at s_t as the number of times the action is taken becomes large.

The decision to exclude requests characteristics from the state signal means that transitions between states could be a result of other factors. We refer to such transitions as γ-transitions (see, for example, Figure 7). We note that such transitions are more likely to originate from "acceptable" states since most of these states would have no actions as-

sociated with them. For example, a sudden increase in the number of clients requests may cause a violation in CPU utilization; i.e., a γ-transition from an "acceptable" state with no CPU utilization violation to a "violation" state. It may also be possible for such transitions to originate from "violation" states. Revisiting our example in Table 2, it might be that all three actions of the policy of Figure 3 are invalid, in which case no action could be taken while the system is in state s_1. The only possible transition, in this case, would be a γ-transition. We note, however, that such transitions are rare in comparison to those originating from "acceptable" states since it is unlikely that all state actions would fail within a single management interval. The existence of γ-transitions in the state-transition graph introduces some interesting challenges for the learning agent. First, the agent may have to decide whether doing nothing (i.e., taking a γ-action) while in state s might be better than, say, taking an action advocated by the violated policies. This may require having to learn the action-values associated with the γ-transitions (i.e., $Q_t(s, \gamma)$). Second, the learning agent may need to distinguish between two "acceptable" states if past experience shows that one state is more unstable than another. The measure of stability could be based on the characteristics of γ-transitions.

9.2 Future Work

Policy conflicts remain one of, if not, the most challenging area in policy-driven autonomic management. On the one hand, conflicts due to policy overlaps can, in most cases, be detected and corrected by analyzing static policy characteristics. On the other hand, policy conflicts which arise from dynamic characteristics specific to policy interactions can only be detected at run-time. For autonomic systems to function correctly, these kinds of conflicts need to be addressed. To what extent Reinforcement Learning could help address some of these challenges is something we hope to address in our future work.

Model-based Reinforcement Learning methods tend to be computationally demanding, even for fairly small state spaces, and could be costly when implemented in a live system. As pointed out previously, this is often due to the size of the state space as well as the computations required to process information associated with the states and actions. The key challenge then is ensuring that computational costs specific to on-line learning tasks do not hinder the learning process. In order to address this challenge, we have begun looking at how *management policies* (see Section 3.2.3) could be used to optimize resources usage during the learning process. This may include, for example, deciding on the circumstances under which computation-intensive algorithms (i.e., action-value estimations) could be executed or paused depending on the current behavior of the system. We are also interested in the use of management policies for "tuning" the behavior of algorithms to meet the resource constraints imposed by the environment. This may, for example, involve dynamically selecting the types of *updates* to be performed in order to minimize the algorithms' use of computational resources. For instance, management policies could be used to determine a reasonable value for k (which determines how many updates can be performed) in the Dyna-Q algorithm (see Algorithm 1 in Section 2).

The use of policies in autonomic computing means that the system must be able to adapt not only to how it uses the policies, but also to run-time policy modifications. In the context of where policies are used to drive autonomic management, this often means dynamically changing the parameters of the policies, enabling/disabling policies or actions within policies, or adding new policies onto an active set of policies. A key question then is whether a model "learned" from the use of one set of policies could be applied to another set of "similar" policies, or whether a new model must be learned from scratch as a result of run-time changes to the policies driving autonomic management. Our most recent work [31] has began addressing some of the questions.

References

[1] R. Murch, *Autonomic Computing*. IBM Press., 2004.

[2] R. M. Bahati, M. A. Bauer, and E. M. Vieira, "Adaptation Stratergies in Policy-Driven Autonomic Management," in *International Conference on Autonomic and Autonomous Systems (ICAS'07)*, Athens, Greece, July 2007, p. 16.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an Introduction*. MIT Press, 1998.

[4] L. P. Kaelbing, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," in *Journal of Artificial Intelligence Research*, April 1996, pp. 237–285.

[5] R. S. Sutton, "Integrated Architecture for Learning, Planning, and Reacting based on Approximating Dynamic Programming," in *International Conference on Machine Learning*, Austin, TX, USA, 1990, pp. 216–224.

[6] A. W. Moore and C. G. Atkeson, "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time," in *Machine Learning*, vol. 13, no. 1, October 1993, pp. 103–130.

[7] J. Ping and R. J. Williams, "Efficient Learning and Planning Within the Dyna Framework," in *International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1993, pp. 281–290.

[8] R. S. Sutton, "Dyna, an Integrated Architecture for Learning, Planning, and Reacting," in *SIGART Bulletin*, vol. 2, no. 4, 1991.

[9] N. Damianou, N. Dulay, E. C. Lupu, and M. S. Sloman, "Ponder: A Language for Specifying Security and Management Policies for Distributed Systems: The Language Specification," Technical Report, Imperial College, London, UK, Version 2.1, April 2000.

[10] H. L. Lutfiyya, G. Molenkamp, M. J. Katchabaw, and M. A. Bauer, "Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-based Framework," in *International Workshop on Policies for Distributed Systems and Networks (POLICY'01)*, Bristol, UK, January 2001, pp. 185–201.

[11] J. O. Kephart and W. E. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies," in *IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, 2004, pp. 3–12.

[12] S. Wang, D. Xuan, R. Bettati, and W. Zhao, "Providing Absolute Differentiated Services for Real-Time Applications in Static-Priority Scheduling Networks," in *IEEE/ACM Transactions on Networking (TON'04)*, vol. 2, December 2004, pp. 326–339.

[13] T. Kelly, "Utility-directed Allocation," in *Workhop on Algorithms and Architectures for Self-Managing Systems*, San Diego, CA, USA, June 2003.

[14] P. Thomas, D. Teneketzis, and J. K. MacKie-Mason, "A Market-based Approach to Optimal Resource Allocation in Integrated-Services Connection-Oriented Networks," in *INFORMS Telecommunications Conference*, Boca Raton, FL, USA, 2000.

[15] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility Functions in Autonomic Systems," in *International Conference on Autonomic Computing (ICAC'04)*, New York, NY, USA, May 2004, pp. 70–77.

[16] M. J. Katchabaw, "Quality of Service Resource Management," Ph.D. dissertation, The University of Western Ontario, London, ON, Canada, June 2002.

[17] R. M. Bahati, M. A. Bauer, C. Ahn, O. K. Baek, and E. M. Vieira, "Policy-based Autonomic Management of an Apache Web Server," in *International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'06)*, vol. 2, no. 1, Erfurt, Germany, September 2006, pp. 21–30.

[18] R. M. Bahati, M. A. Bauer, and E. M. Vieira, "Policy-driven Autonomic Management of Multi-component Systems," in *IBM International Conference on Computer Science and Software Engineering (CASCON'07)*, Richmod Hill, ON, Canada, October 2007, pp. 137–151.

[19] R. Das, G. Tesauro, and W. E. Walsh, "Model-Based and Model-Free Approaches to Autonomic Resource Allocation," Technical Report, IBM Research," RC23802, 2005.

[20] G. Tesauro, "Online Resource Allocation Using Decompositional Reinforcement Learning," in *Association for the Advancement of Artificial Intelligence (AAAI'05)*, 2005.

[21] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," in *International Conference on Autonomic Computing (ICAC'06)*, Dublin, Ireland, June 2006, pp. 65–73.

[22] D. Vengerov and N. Iakovlev, "A Reinforcement Learning Framework for Dynamic Resource Allocation: First Results," in *International Conference on Autonomic Computing (ICAC'05)*, Seattle, WA, USA, January 2005, pp. 339–340.

[23] P. Vienne and J. Sourrouille, "A Middleware for Autonomic QoS Management based on Learning," in *International Conference on Sofware Engineering and Middleware*, Lisbon, Portugal, September 2005, pp. 1–8.

[24] Apache Http Server Project. [Online]. Available: http://www.apache.org/

[25] PHP. [Online]. Available: http://www.php.net/

[26] MySQL Database. [Online]. Available: http://www.mysql.com/

[27] PHP Bulletin Board. [Online]. Available: http://www.phpbb.com/

[28] RFC2616: Hypertext Transfer Protocol – HTTP/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.txt.

[29] eAccelerator. [Online]. Available: http://eaccelerator.net/

[30] Linux. [Online]. Available: http://www.linux.org/

[31] R. M. Bahati and M. A. Bauer, "Adapting to Run-time Changes in Policies Driving Autonomic Management," in *International Conference on Autonomic and Autonomous Systems (ICAS'08)*, Gosier, Guadeloupe, March 2008, pp. 88–93.

Preliminary 2009 Conference Schedule

http://www.iaria.org/conferences.html

NetWare 2009: June 14-19, 2009 - Athens, Greece

- SENSORCOMM 2009, The Third International Conference on Sensor Technologies and Applications
- SECURWARE 2009, The Third International Conference on Emerging Security Information, Systems and Technologies
- MESH 2009, The Second International Conference on Advances in Mesh Networks
- AFIN 2009, The First International Conference on Advances in Future Internet
- DEPEND 2009, The Second International Conference on Dependability

NexComm 2009: July 19-24, 2009 - Colmar, France

- CTRQ 2009, The Second International Conference on Communication Theory, Reliability, and Quality of Service
- ICDT 2009, The Fourth International Conference on Digital Telecommunications
- SPACOMM 2009, The First International Conference on Advances in Satellite and Space Communications
- MMEDIA 2009, The First International Conferences on Advances in Multimedia

InfoWare 2009: August 25-31, 2009 – Cannes, French Riviera, France

- ICCGI 2009, The Fourth International Multi-Conference on Computing in the Global Information Technology
- ICWMC 2009, The Fifth International Conference on Wireless and Mobile Communications
- INTERNET 2009, The First International Conference on Evolving Internet

SoftNet 2009: September 20-25, 2009 - Porto, Portugal

- ICSEA 2009, The Fourth International Conference on Software Engineering Advances
  - SEDES 2009: Simpósio para Estudantes de Doutoramento em Engenharia de Software
- ICSNC 2009, The Fourth International Conference on Systems and Networks Communications
- CENTRIC 2009, The Second International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services
- VALID 2009, The First International Conference on Advances in System Testing and Validation Lifecycle
- SIMUL 2009, The First International Conference on Advances in System Simulation

NexTech 2009: October 11-16, 2009 - Sliema, Malta

- UBICOMM 2009, The Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies
- ADVCOMP 2009, The Third International Conference on Advanced Engineering Computing and Applications in Sciences
- CENICS 2009, The Second International Conference on Advances in Circuits, Electronics and Micro-electronics
- AP2PS 2009, The First International Conference on Advances in P2P Systems
- EMERGING 2009, The First International Conference on Emerging Network Intelligence
- SEMAPRO 2009, The Third International Conference on Advances in Semantic Processing