A publication of ISCA*:
International Society for Computers
and Their Applications

INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

* "International Journal of Computers and Their Applications is abstracted and indexed in INSPEC and Scopus."

Volume 23, No. 2, June 2016 ISSN 1076-5204

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

ISCA Headquarters···•···64 White Oak Court, Winona, MN 55987···•···Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: http://www.isca-hq.org

Guest Editors' Note

CATA (Computers and their Applications) is the flagship conference for the International Society of Computers and their Applications (ISCA). The intent of the conference has been to blend theory and practice as a means of stimulating researchers from both research dimensions. The papers for this special issue have been selected to illustrate the spectrum of the 61 papers presented at the CATA 2016 conference. The authors were asked to extend their paper to make the papers journal appropriate, and to change the title of their paper (to avoid confusion with their conference paper). This CATA special issue contains the following six papers.

In their paper "Performance evaluation technique for computer systems with finite input source", T. Kinoshita *et al.* focuses on a queuing network model, proposing an approximation technique to evaluation the performance of such approach.

In their paper "A pipelined implementation of Hash Stream1-Synthetic Initialization Vector encryption algorithm", M. Durai *et al.* present a hardware accelerator featuring parallel execution of the main key generation phases.

In their paper "Consolidation of data in multiple databases", K. Periyasamy *et al.* discuss the critical problem of database merging and propose a consolidation method based on the combination of a linguistic, structural and constraint matching approach.

In their paper "Designing secure computer systems as purposeful systems", M. Etschmaier *et al.* propose a framework detailing the design and optimization of secure systems. They consider voting machines as a case study.

In their paper "Server mechanisms for guaranteeing schedulability with RTOS processing and improving application responsiveness by slack reclaiming", K. Hasegawa *et al.* focus on the problem of task schedulability assurance in the context of real-time operating systems.

Lastly, in their paper "A unified cloud metering framework", K. Sobh *et al.* describe a novel metering framework for cloud computing to support the economic side of cloud computing and increase its transparency to users. Then, they conduct several experiments using their implementation to evaluate the proposed system.

Antoine BOSSARD and Les MILLER

Performance Evaluation Technique for Computer Systems with Finite Input Source

Toshiyuki Kinoshita*, Matrazali Noorafiza*, and Kaori Katsumata*
Tokyo University of Technology, Tokyo, JAPAN

Abstract

Queuing network techniques are effective for evaluating the performance of computer systems. We discuss a queuing network technique for computer systems in finite input source. The finite number of terminals exists in the network and a job in the network moves to the server that includes CPU, I/O equipment and memory after think-time at the terminal. When the job arrives at the server, it acquires a part of memory and executes CPU and I/O processing in the server. After the job completes CPU and I/O processing, it releases the memory and goes back to its original terminal. However, because the memory resource can be considered as a secondary resource for the CPU and I/O equipment, the queuing network model has no product form solution and cannot be calculated the exact solutions.

We proposed here an approximation queuing network technique for calculating the performance measures of computer systems with finite input source on which multiple types of jobs exist. This technique involves dividing the queuing network into two levels; one is "inner level" in which a job executes CPU and I/O processing, and the other is "outer level" that includes terminals and communication lines. By dividing the network into two levels, we can prevent the number of states of the network from increasing and approximately calculate the performance measures of the network. We evaluated the proposed approximation technique by using numerical experiments and clarified the characteristics of the system response time and the mean number of jobs in the inner level.

Key Words: Queuing network, central server model, finite input source.

General Terms: Computer system performance, performance evaluation, queuing theory.

1 Introduction

Queuing network techniques are effective for evaluating the performance of computer systems. In computer systems, two or more jobs are generally executed at the same time, which causes delays due to conflicts in accessing hardware or software resources such as the CPU, I/O equipment, or data files. We can evaluate how this delay affects the computer system performance by using a queuing network technique. Some queuing networks have an explicit exact solution, which is called a product form solution [1]. With this solution, we can easily calculate the performance measures of computer systems, for example the busy ratio of hardware and the job response time, and so on. However, when the exclusion controls are active or when a memory resource exists, the queuing network does not have the product form solution. To calculate an exact solution of a queuing network that does not have the product form solution, we have to construct a Markov chain that describes the stochastic characteristics of the queuing network and numerically solve its equilibrium equations. When the number of jobs or the amount of hardware in the network increases, the number of states of the queuing network drastically increases. Since the number of unknown quantities in the equilibrium equations is equal to the number of states of the queuing network, the number of unknown quantities in the equilibrium equations also drastically increases. Therefore, we cannot perform calculating the exact solution of the queuing network numerically.

Here, we discuss the queuing network technique for computer systems with finite input source (Figure 1). The finite number of terminals exists in the system and a job is dedicated to its own terminal. After a think-time at the terminal, the job moves to the server and acquires a part of the memory and executes CPU and I/O processing. When the job completes CPU and I/O processing at the server, it releases the memory and goes back to its original terminal.

Since a job executes CPU and I/O processing occupying the memory, the memory can be considered as a secondary resource for the CPU and I/O equipment. Generally, when a queuing network includes a secondary resource, it does not have product form solutions. To have the strict solution of the model, we have to construct a Markov chain which describes the entire model and have to numerically solve its equilibrium equations. In order to prevent the number of states of the Markov chain from increasing, we divide the model into two levels, one is outer level that includes the terminals and communication lines, and the other is inner level that includes CPU, I/O equipment and memory resources. Similarly, in [5, 9, 11], two different types of jobs exist in the network. Both job behavior in the inner level and the outer level differs for each job class. Although both the inner level and the outer level has a

* 1404-1 Katakura Hachioji. Email: kinoshi@stf.teu.ac.jp, noorafiza.matrazali@gmail.com, and c0113134d2@edu.teu.ac.jp.

product form solution when the model has a single job class, the inner level does not have a product form solution when the model has multiple job classes. Therefore, an approximation is needed to analyze the inner level.

In this paper, we have proposed an approximation technique for calculating the performance measures of computer systems with finite input source by using the queuing network technique. We previously reported the results for the finite input source model when all jobs acquire the same units of memory [3]. In this study, we have reported when a job in the difference job class acquires the difference units of memory.

Dividing the model into two levels is one of two-layer queuing network techniques [7, 10]. Our proposed technique is also a two-layer technique for computer systems with finite input source.

In our previous study [4], we reported an approximation technique for evaluating performance of computer systems with file resources. Meanwhile, heterogeneous parallel computer systems with distributed memory was researched in [8], and the Markov chain involving two-dimensional state transition similar to our proposed model was discussed in [2].

2 Model Description

The CPU and I/O model in the inner level is equivalent to the ordinary central server model with multiple job types (each of which is called a job class). In this model, R job classes exist, and each of them is numbered r = 1, 2, ..., R by affixing r. We denote n_r as the numbers of jobs of job class r in the central server model and n as the total number of jobs in the central server model. We also denote i_r as the number of jobs of job class r in the inner level (the difference of n_r and i_r is the number of jobs in the system waiting queue). The inner level consists of single CPU node and multiple I/O nodes. We denote M as the number of I/O nodes. The I/O nodes are numbered m = 1, 2, ..., M by affixing m, and the CPU node is numbered m = 0 by also affixing m. The service rate of job class r at the CPU node is μ_{0r}, and the service rate of job class r at an I/O node m is μ_{mr}. The service time at each node is a mutually independent random variable subject to common exponential distributions. Jobs are scheduled on a first come first served (FCFS) principle at all nodes. In real system, the priority scheduling is often used at the CPU node. However, since the queuing system that includes the priority scheduled node has no product form solution, we assume FCFS principle even at the CPU node. At the end of CPU processing, a job probabilistically selects an I/O node and moves to it, or completes CPU and I/O processing and goes back to the terminal. The selection probability of I/O node m of job class r is p_m^r (m = 1, 2, ..., M; r = 1, 2, ..., R) and the completion probability of job class r is p_0^r. Therefore, $\sum_{m=0}^{M} p_m^r = 1$ (r = 1, 2, ..., R).

In the outer level, there are Kr terminals of job class r (r = 1, 2, ..., R) and one job is dedicated at each terminal. The job stays in the terminal for short while. The staying time is called "think-time". The think-time is mutually independent random variable subject too common exponential distribution with parameter v_r of job class r (v_r is job departure rate from the terminal).

Memory resources are added to this central server model (Figure 1). We denote as the number of the units of the memory acquired by a job of job class r and S as the total number of the units of the memory. After the think-time, a job of job class r moves to the inner level, and requests and acquires units of the memory before entering the central server model. If sufficient units of the memory do not exist, the job joins the system waiting queue and waits for the memory to be released by another job. When the job completes CPU and I/O processing, it releases the memory and leaves the inner level and goes back to its own terminal in the outer level. Since the job has to acquire the memory before entering the central server model, the total number of units of the occupied memory in the central server model has to be less than or equal to S, i.e. $\sum_{r=1}^{R} n_r S_r \le S$. Moreover we assume $S \le K_r S_r$ (r = 1, 2, ..., R).

By replacing "CPU → outer level transition" with "CPU → CPU transition," the central server model is modified to a closed model in which the number of jobs is constant (Figure 2). In this modified model, when "CPU → CPU transition" occurs, the job terminates and a new job is born. Therefore, the mean job response time is the mean time between two successive "CPU → CPU transitions." This means that the job response time can be considered as the job lifetime.

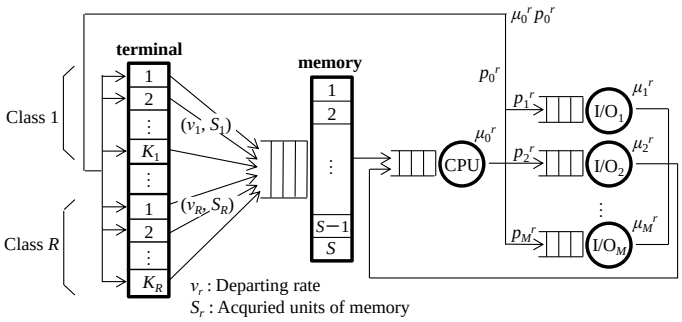


Figure 1: Finite source model with multi job class

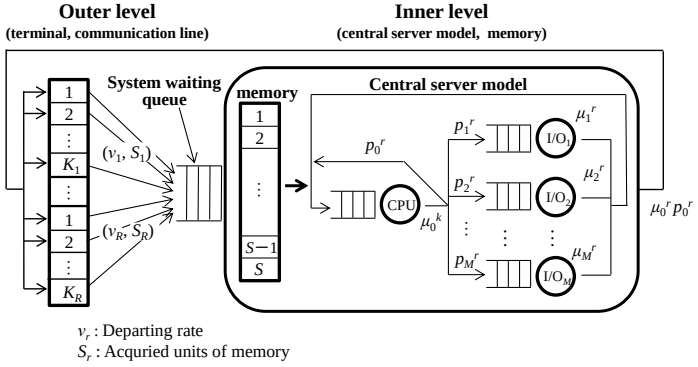


Figure 2: Concept of approximation

3 Approximation Model

To obtain the exact solution of the central server model with finite input source, we have to describe the entire model with a single Markov chain for each job class. However, this causes the number of states of the Markov chain to drastically increase when the number of jobs or the number of nodes in the network increases. By dividing the network into two levels, and describing each level with two Markov chains, we can prevent the number of states of the model from increasing (Figure 2). We use the following notations.

t_r : mean think-time of job class r
v_r : departure rate from the terminal of job class r
τ_{rm} : mean total service time of job class r at node-m
S_r : number of units of memory acquired by a job of job class r
S : total number of units of memory resorce
n_{rm} : number of jobs of job class r at node-m
(r =1, 2, ..., R; m =0, 1, ..., M)
K_r : number of jobs of job class r
(= number of terminals of job class r)
i_r : number of jobs of job class r in the inner level
n_r : number of jobs of job class r in the central server model
$\boldsymbol{n} = (n_1, n_2, ..., n_R)$
: vector of number of jobs in the central server model (n_r = 1, 2, ..., K_r)
$\boldsymbol{n}^* = (n_{10}, n_{11}, ..., n_{1M}, n_{20}, n_{21}, ..., n_{2M}, ..., n_{R0}, n_{R1}, ..., n_{RM})$
$\boldsymbol{F}(\boldsymbol{n}) = \{\boldsymbol{n}^* \mid \sum_{m=0}^{M} n_{rm} = n_r, n_{rm} \geq 0 \ (m=0, 1, ..., M)\}$

$(n_1S_1 + n_2S_2 + ... + n_RS_R \leqq S)$

: set of all feasible states of the central server model when the number of jobs of job class r is n_r
$P_s(\boldsymbol{n}^*)$: steady-state probability of state $\boldsymbol{n}^*$
$T_{\boldsymbol{n}}{}^r$: mean job response time of the central server model of job class r when the vector of number of jobs is $\boldsymbol{n}$
$\mu_{\boldsymbol{n}}{}^r$: mean job service rate from the central server model of job class r when the vector of number of jobs is $\boldsymbol{n}$
T^r : system response time of job class r (= lifetime of job class r)

Since the central server model in the inner level is equivalent to the ordinary central server model with multiple job classes, it has the product form solution. Then the steady-state probability $P_s(\boldsymbol{n}^*)$ is represented by the following formula [1, 6].

$$P_s(\boldsymbol{n}^*) = \frac{\prod_{r=1}^{R}\prod_{m=0}^{M} \tau_{rm}{}^{n_{rm}}}{\varphi(n_1, n_2, \cdots, n_R, M)}$$

where $\varphi(n_1, n_2, \cdots, n_R, M) = \sum_{n \in F(\boldsymbol{n})} \prod_{r=1}^{R}\prod_{m=0}^{M} \tau_{rm}{}^{n_{rm}}$ is the normalizing constant of steady-state probabil-ities when the number of jobs of job class r in the central server model is n_r (=0, 1, 2, …, K_r; r =1, 2, ..., R). From these steady-state probabilities, we can calculate the mean job response time $T_{\boldsymbol{n}}{}^r$ of job class r as follows when the number of jobs in the central server model is n_r.

$$T_{\boldsymbol{n}}{}^r = \frac{n_r \cdot \varphi(n_1, \cdots, n_r, \cdots, n_R, M)}{\varphi(n_1, \cdots, n_r - 1, \cdots, n_R, M)}$$

The memory resource in our model can be considered as an M/M/S queuing model with S servers. In an ordinary M/M/S queuing model, the service rate at a server is constant, regardless of the number of guests in the service. In the memory resource of our model, however, the service rate changes depending on the number of occupied memories. The mean job response time $T_{\boldsymbol{n}}{}^r$ of job class r (=1, 2, ..., R) when the vector of number of jobs is $\boldsymbol{n} = (n_1, n_2, ..., n_R)$ is equal to the mean time while the memory is occupied. Since the service rate of job class r from the central server model $\mu_{\boldsymbol{n}}{}^r$ is denoted as $\mu_{\boldsymbol{n}}^r = \frac{1}{T_{\boldsymbol{n}}^r}$ $\mu_{\boldsymbol{n}}{}^r$ also depends on $\boldsymbol{n} = (n_1, n_2, ..., n_R)$, that is the number of jobs in the central server model. The state transition in the central server model. The state transition of the M/M/S queuing model with two job classes is shown in Figure 3, where the service rates from the central server model change depending on the number of jobs in the central server model. This is a two-dimensional birth-death process. The equilibrium equations with the steady-state probability $Q_S(i_1, i_2)$, when the total number of the units of the memory is S and the number of jobs in the inner level is (i_1, i_2), are as follows (similar to the case with higher dimensions). Where s_1 is the maximum integer such as $s_rS_r \leq S$ i.e. $s_r = [S/S_r]$.

(1) $i_1=0, i_2=0$
$(K_1v_1 + K_2v_2) \cdot Q_S(0, 0) = \mu_{10}{}^1 \cdot Q_S(1, 0) + \mu_{01}{}^2 \cdot Q_S(0, 1)$

(2) $0 < i_1S_1 \leq S, i_2 = 0$
$\{(K_1 - i_1)v_1 + K_2v_2 + i_1\mu_{i_1 0}^1\} \cdot Q_S(i_1, 0)$
$= (K_1 - i_1 + 1)v_1 \cdot Q_S(i_1 - 1, 0) + (i_1 + 1)\mu_{i_1+1,0}^1 \cdot Q_S(i_1 + 1, 0)$
$+ \mu_{i_1 1}^2 \cdot Q_S(i_1, 1)$

(3) $S < i_1S_1, i_1 < K_1, i_2 = 0$
$\{(K_1 - i_1)v_1 + K_2v_2 + s_1\mu_{s_1 0}^1\} \cdot Q_S(i_1, 0)$
$= (K_1 - i_1 + 1)v_1 \cdot Q_S(i_1 - 1, 0) + s_1\mu_{s_1 0}^1 \cdot Q_S(i_1 + 1, 0) +$
$\mu_{i_1 1}^2 \cdot Q_S(i_1, 1)$

(4) $i_1 = K_1, i_2 = 0$
$\{K_2v_2 + s_1\mu_{s_1 0}^1\} \cdot Q_S(K_1, 0)$
$= v_1 \cdot Q_S(K_1 - 1, 0) + \mu_{K_1 1}^2 \cdot Q_S(K_1, 1)$

(5) $i_1 = 0, 0 < i_2S_2 \leq S$
$\{K_1v_1 + (K_2 - i_2)v_2 + n_2\mu_{0 i_2}^2\} \cdot Q_S(0, i_2)$
$= (K_2 - i_2 + 1)v_2 \cdot Q_S(0, i_2 - 1) + \mu_{1 i_2}^1 \cdot Q_S(1, i_2) +$
$(i_2 + 1)\mu_{0 i_2 + 1}^2 \cdot Q_S(0, i_2 + 1)$

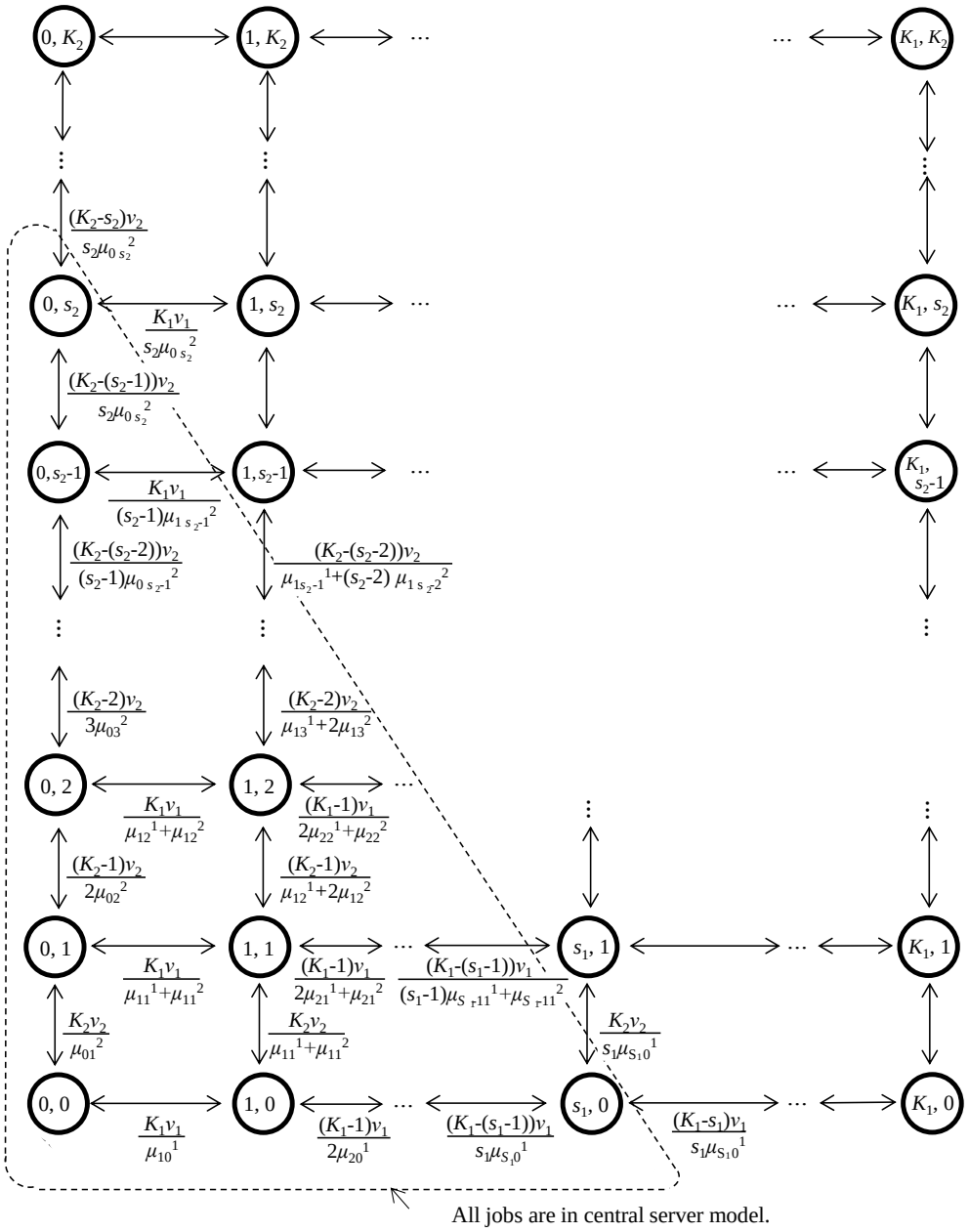


Figure 3: State transition diagram (two job classes)

(6) $i_1 = 0, S < i_2 S_2, i_2 < K_2$

$$\{K_1 v_1 + (K_2 - i_2)\, v_2 + s_2 \mu_{0 S_2}^2\} \cdot Q_S(0, i_2)$$

$$= (K_2 - i_2 + 1)\, v_2 \cdot Q_S(0, i_2 - 1) + \mu_{1 i_2}^1 \cdot Q_S(1, i_2)$$

$$+ s_2 \mu_{0 s_2}^2 \cdot Q_S(0, i_2 + 1)$$

(7) $i_1 = 0, i_2 = K_2$

$$\{K_1 v_1 + s_2 \mu_{0 s_2}^2\} \cdot Q_S(0, K_2)$$

$$= v_2 \cdot Q_S(0, K_2 - 1) + \mu_{1 K_2}^1 \cdot Q_S(1, K_2)$$

(8) $0 < i_1, 0 < i_2, i_1 S_1 + i_2 S_2 \le S - S_1 - S_2$

$$\{(K_1 - i_1)\, v_1 + (K_2 - i_2)\, v_2 + i_1 \mu_{i_1 i_2}^1 + i_2 \mu_{i_1 i_2}^2\} \cdot Q_S(i_1, i_2)$$

$$= (K_1 - i_1 + 1) v_1 \cdot Q_S(i_1 - 1, i_2) + (K_2 i_2 + 1) v_2 \cdot Q_S(i_1, i_2 - 1)$$

$$+ = (i_1 + 1) \mu_{i_1 + 1 i_2}^1 \cdot Q_S(i_1 + 1, i_2) + (i_2 + 1) \mu_{i_1 i_2 + 1}^2 \cdot Q_S(i_1, i_2 + 1)$$

(9) $0 < i_1, 0 < i_2, S - S_1 < i_1 S_1 + i_2 S_2 \le S$

$$\{(K_1 - i_1)\, v_1 + (K_2 - i_2)\, v_2 + i_1 \mu_{i_1 i_2}^1 + i_2 \mu_{i_1 i_2}^2\} \cdot Q_S(i_1, i_2)$$

$$= (K_1 - i_1 + 1) v_1 \cdot Q_S(i_1 - 1, i_2) + (K_2 - i_2 + 1) v_2 \cdot Q_S(i_1, i_2 - 1) +$$

$$i_1 \mu_{i_1 i_2}^1 \cdot Q_S(i_1 + 1, i_2) + (i_2 + 1) \mu_{i_1 i_2}^2 \cdot Q_S(i_1, i_2 + 1)$$

(10) $0 < i_1, 0 < i_2, S - S_2 < i_1S_1 + i_2S_2 \le S$

$$\{(K_1 - i_1)\,v_1 + (K_2 - i_2)\,v_2 + i_1\,\mu^1_{i_1 i_2} + i_2\,\mu^2_{i_1 i_2}\} \cdot Q_S(i_1, i_2)$$
$$= (K_1 - i_1 + 1)\,v_1 \cdot Q_S(i_1 - 1, i_2) + (K_2 - i_2 + 1)\,v_2 \cdot Q_S(i_1, i_2 - 1) + (i_1 + 1)\,\mu^1_{i_1 i_2} \cdot Q_S(i_1 + 1, i_2) + i_2\,\mu^2_{i_1 i_2} \cdot Q_S(i_1, i_2 + 1)$$

(11) $S < i_1S_1 + i_2S_2,\ 0 < i_1 \le K_1,\ 0 < i_2 \le K_2$

$\Re(i_1, i_2)$ denotes the set of the shortest routes from (0, 0) to (i_1, i_2), and there is the lattice point (j_1, j_2) on a route $u \in \Re(i_1, i_2)$ such as $j_1S_1 + j_2S_2 \le S < (j_1 + 1)\,S_1 + j_2S_2$ or $j_1S_1 + j_2S_2 \le S < j_1S_j + (j_2 + 1)\,S_2$. When we denote the steady-state probability along the route u as $Q_S^{j_1 j_2}(i_1, i_2)$, the following equilibrium equation is held.

$$\{(K_1 - i_1)\nu_1 + (K_2 - i_2)\nu_2 + j_1\mu^1_{i_1 i_2} + i_2\mu^2_{i_1 i_2}\} \cdot Q_S^{j_1 j_2}(i_1, i_2)$$
$$= (K_1 - i_1 + 1)\nu_1 \cdot Q_S^{j_1 j_2}(i_1 - 1, i_2) + (K_2 - i_2 + 1)\nu_2 \cdot Q_S^{j_1 j_2}(i_1, i_2 - 1) + j_1\mu^1_{j_1 j_2} \cdot Q_S^{j_1 j_2}(i_1 + 1, i_2) + j_2\mu^2_{j_1 j_2} \cdot Q_S^{j_1 j_2}(i_1, i_2 + 1)$$

$Q_S(i_1, i_2)$ can be represented as follows.

$$Q_S(i_1, i_2) = \sum_{\substack{u \in \Re(i_1, i_2) \\ (j_1, j_2)\ is\ on\ u}} Q_S^{j_1 j_2}(i_1, i_2)$$

For the state (i_1, i_2) of the Markov chain, when $i_1S_1 + i_2S_2 \le S$, all jobs are in the central server model and executing CPU and I/O processing, and when $S < i_1S_1 + i_2S_2$ some jobs are in the system waiting queue and waiting for a part of the memory to be released. The transition diagram of the two-dimensional birth-death process is shown in Figure 3. However, the equilibrium equations do not have the product form solution. Therefore, some approximation is required to solve it.

When the model has a single job class, it can be described with a one-dimensional birth-death process. Its transition diagram is shown in Figure 4, and the equilibrium equations are as follows:

$$K_1\,v_1 \cdot Q_S(0) = \mu_1{}^1 \cdot Q_S(1)$$
$$\{(K_1 - i_1)\,v_1 + i_1\,\mu^1_{i_1}\} \cdot Q_S(i_1) = (K_1 - i_1 + 1)\,v_1 \cdot Q_S(i_1 - 1) + (i_1 + 1)\,\mu^1_{i_1 + 1} \cdot Q_S(i_1 + 1) \quad (0 < i_1S_1 \le S)$$

$$\{(K_1 - i_1)\,v_1 + s_1\,\mu^1_{s_1}\} \cdot Q_S(i_1) = (K_1 - i_1 + 1)\,v_1 \cdot Q_S(i_1 - 1) + s_1\,\mu^1_{s_1} \cdot Q_S(i_1 + 1) \quad (S < i_1S_1,\ i_1 < K_1)$$

$$s_1\mu^1_{s_1} \cdot Q_S(K_1) = v_1 \cdot Q_S(K_1 - 1)$$

The solutions for the equilibrium equations are described in the following product form.

$$Q_S(i_1) = \begin{cases} Q_S(0) \cdot \dfrac{K_1\nu_1}{1 \cdot \mu^1_1} \cdot \dfrac{(K_1 - 1)\nu_1}{2 \cdot \mu^1_2} \cdots \dfrac{(K_1 - i_1 + 1)\nu_1}{i_1 \cdot \mu^1_{i_1}} & (i_1 = 1, 2, \ldots, s_1) \\ Q_S(0) \cdot \displaystyle\prod_{i=1}^{s_1} \frac{(K_1 - i + 1)\nu_1}{i \cdot \mu^1_i} \times \frac{(K_1 - s_1)\nu_1}{s_1 \cdot \mu^1_{s_1}} \cdot \frac{(K_1 - s_1 + 1)\nu_1}{s_1 \cdot \mu^1_{s_1}} & \\ \quad \cdots\cdots \dfrac{(K_1 - i_1 + 1)\nu_1}{s_1 \cdot \mu^1_{s_1}} & (i_1 = s_1 + 1, \ldots, K_1) \end{cases}$$

In this formula, for the state transition at i =1, 2, ..., $s_1 - 1$, multiply by factor $\frac{(K_1 - i + 1)\nu_1}{i \cdot \mu^1_i}$, while for the state transition at i = s_1, s_1+1, ..., K_1 multiply by factor $\frac{(K_1 - i + 1)\nu_1}{s_1 \cdot \mu^1_{s_1}}$. For two-dimension case, we consider a route from lattice point (0, 0) to (i_1, i_2) shown in Figure 5, and for the horizontal state transition at the lattice point (i_1, i_2) such as $i_1S_1 + i_2S_2 \le S$ on the route, multiply by factor $\frac{(K_1 - i_1 + 1)\nu_1}{i_1 \cdot \mu^1_{i_1 i_2} + i_2 \cdot \mu^2_{i_1 i_2}}$, and multiply by factor $\frac{(K_1 - i_1 + 1)\nu_1}{i_1 \cdot \mu^1_{i_1 i_2} + i_2 \cdot \mu^2_{i_1 i_2}}$, and multiply by factor $\frac{(K_2 - i_2 + 1)\nu_2}{i_1 \cdot \mu^1_{i_1 i_2} + i_2 \cdot \mu^2_{i_1 i_2}}$ for the vertical state transition. When the lattice point (i_1, i_2) such as $S < i_1S_1 + i_2S_2$, for the state transition outside of the lattice point (j_1, j_2) such as $S - S_1 < j_1S_1 + j_2S_2 \le S$ or $S - S_2 < j_1S_1 + j_2S_2 \le S$ on the route (between (j_1, j_2) and (i_1, i_2)), multiply by factor $\frac{(K_1 - j_1 + 1)\nu_1}{j_1 \cdot \mu^1_{j_1 j_2} + j_2 \cdot \mu^2_{j_1 j_2}}$ for the horizontal state transition or $\frac{(K_2 - j_2 + 1)\nu_2}{j_1 \cdot \mu^1_{j_1 j_2} + j_2 \cdot \mu^2_{j_1 j_2}}$ for the vertical state transition. Thus, the coefficient of $Q_S(i_1, i_2)$ related to $Q_S(0, 0)$ is represented as the summation of the product along all the routes from (0, 0) to (i_1, i_2). For example, for the route from (0, 0) to (1, 2) when S=4, S_1=2, S_2=1, and K_1=5, K_2=4, which is

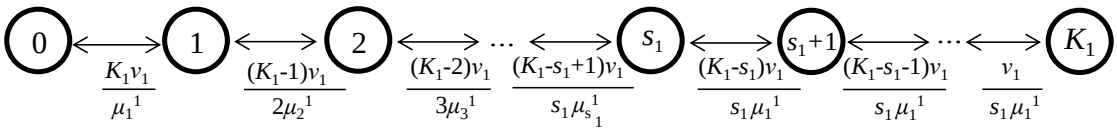


Figure 4: State transition diagram (single job class)

the case of $i_1S_1 + i_2S_2 \le S\ (2i_1 + i_2 \le 4)$, the product along the route of broken line (i) in Figure 5 is $Q_S(0,0) \cdot \frac{5\nu_2}{\mu_{01}^2} \cdot \frac{4\nu_2}{2\mu_{02}^2} \cdot \frac{4\nu_1}{\mu_{12}^1 + 2\mu_{12}^2}$. For the route from (0, 0) to (3, 2), which is the case of $S < i_1S_1 + i_2S_2\ (4 < 2i_1 + i_2)$, the multiplication along the route (ii) is $Q_S(0,0) \cdot \frac{4\nu_1}{\mu_{01}^2} \times \frac{5\nu_2}{\mu_{11}^1 + \mu_{11}^2} \cdot \frac{3\nu_1}{\mu_{11}^1 + \mu_{11}^2} \cdot \frac{4\nu_2}{\mu_{11}^1 + \mu_{11}^2} \cdot \frac{2\nu_1}{\mu_{11}^1 + \mu_{11}^2}$.

Since there are multiple routes from (0, 0) to (i_1, i_2), the coefficient of $Q_S(i_1, i_2)$ related to $Q_S(0, 0)$ is approximately represented as the total of the product along all routes. Similarly, to the case above, we can approximately calculate the state probability of a queuing network with multiple job classes when R >2.

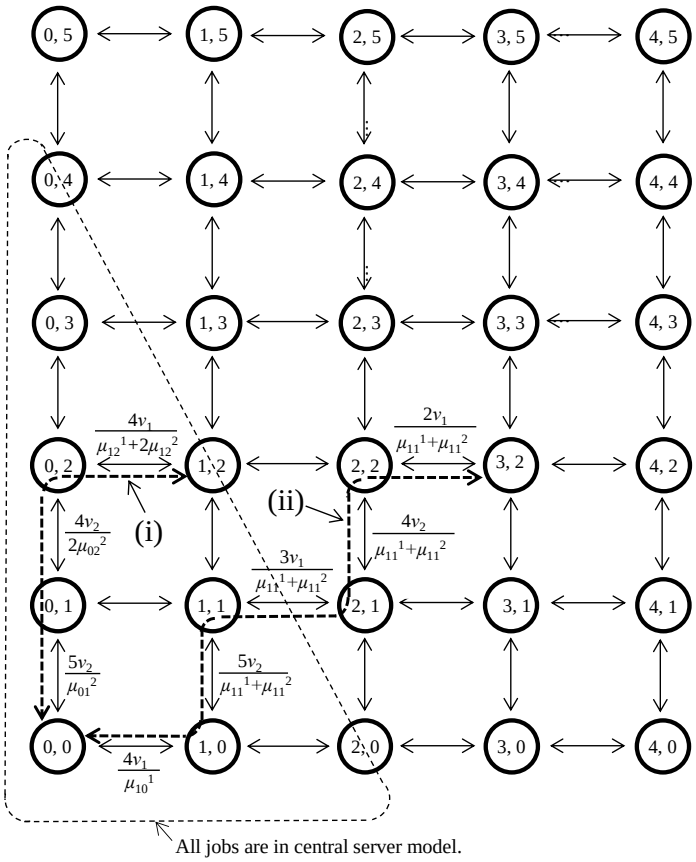


Figure 5: Calculating state probability for two job classes (K_1=4, K_2=5, S=4, S_1=2, S_2=1, s_1=2, s_2=4)

4 Numerical Experiments

We evaluated the proposed approximation technique through numerical experiments. In the experiments, we focus on the mean system response time and the mean number of jobs because they most clearly show the characteristics of the model and the practical results. We used the following parameters.

(1) Number of terminals: K_1 = 3, 4, …, 20; K_2 = 3
(2) Number of memory resources: S = 3
(3) Think-time: (t_1, t_2) = (20, 10), (5.0, 2.5),
where t_r is the think-time of job class r (r = 1, 2).
(4) Number of I/O nodes: M = 2
(5) Total service time at each node
τ_{10}=1.0, τ_{11}=τ_{12}=1.0
τ_{20}=1.0, τ_{21}=τ_{22}=0.5,

where τ_{rm} is the total service time of job class r at node m.

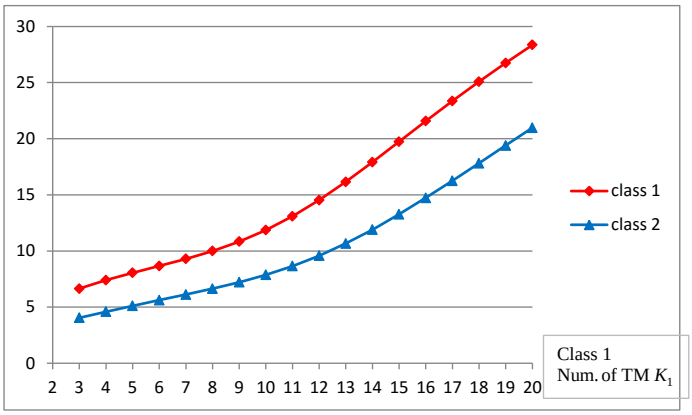
Figures 6 show the mean system response times and mean num-bers of jobs in the inner level of job classes 1 and 2 respectively, when K_2 is fixed at 3, and K_1 changes from 3 to 20. The mean system response time is the mean time from job arrival to departure from the inner level (that is the mean time from departure from the terminal to coming back to the terminal). Similarly, to the case of a single job class, the mean system response time for both job class increases monotonically in a convex curve. As the number of terminals K_1 of job class 1 is increased and K_2 is fixed (the only traffic of job class 1 is increased), both the mean response time of job class 1 and job class 2 increases, because the entire central server model is more crowded by increasing the traffic of the job class 1. We can see that the mean response time of job class 1 and job class 2 is nearly increasing in the range of heavier traffic. This reason can be presumed that the behavior of the mean system response time in the heavier traffic range is approximately linear to the number of jobs.

In Figure 7, total service times are set as follows:

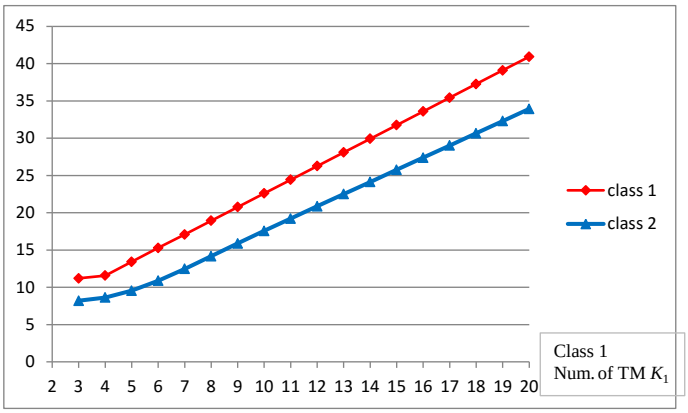
τ_{10}=1.0, τ_{11}=τ_{12}=1.0
τ_{20}=1.0, τ_{21}=τ_{22}=1.0,

This means that jobs in job class 1 and 2 have the same behavior in the central server model in the inner level and have different think-time, t_1 and t_2, that are constant 20 and 10 respectively. We can see in Figure 7(a) that the system response times of jobs in job class 1 and 2 are quite similar when the number of terminals K_1 of job class 1 changes from 3 to 20. It is natural because we assume that job behavior of job class 1 and 2 in the central server model is same. On the other hand, the mean numbers of jobs in inner model (these are denoted as i_1, i_2) are quite different be-tween job class 1 and 2. That is because since the think-time t_1 of job class 1 is much larger than that of job class 2, the jobs in job class 1 stay in the terminal longer than job class 2 and the number of jobs of job class 1 in the inner level is smaller than that of job class 2 even though the total number of terminals of job class 1 is less than that of job class 2.

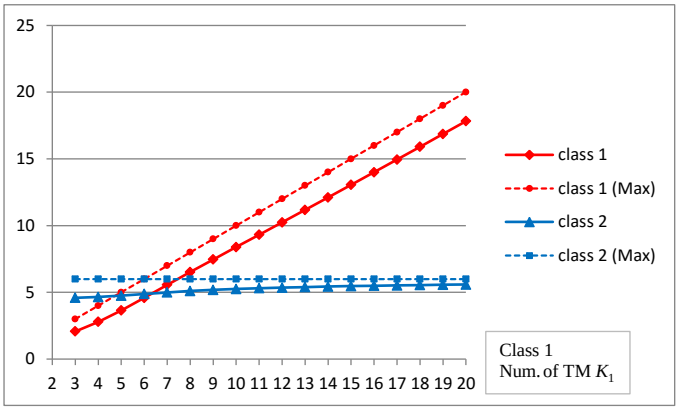
Figure 7(c)-(d) show change of job behavior when the think-time t_1 of job class 1 changes from 2 to 20 (the think-time t_2 of job class 2 is still constant). The Figure 7(c) shows that the system response times of job class 1 and 2 are also similar since the behavior of job class 1 and 2 in the central server



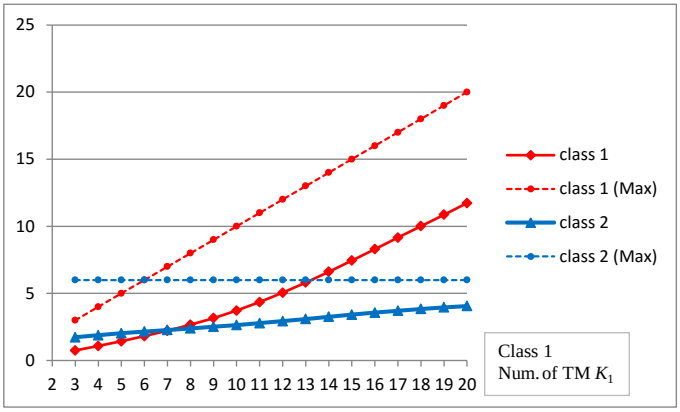
(a) System response time
(K_2=3, (t_1, t_2)=(20, 10))



(c) System response time
(K_2=3, (t_1, t_2)=(5, 2.5)



(b) Mean number of jobs in inner level
(K_2=3, (t_1, t_2)=(20, 10))



(d) Mean number of jobs in inner level
(K_2=3, (t_1, t_2)=(5, 2.5))

Figure 6: Experimental results
(τ_{10}=1.0, τ_{11}=τ_{12}=1.0; τ_{20}=1.0, τ_{21}=τ_{22}=0.5)

model are the same. Moreover, Figure 7(d) shows that both the numbers of jobs of job class 1 and 2 in inner level decrease when t_1 becomes longer.

5 Conclusion

We proposed an approximation technique for evaluating the performance of computer systems in finite input source using a queuing network technique and analyzed its performance measures through numerical experiments. The concept of the approximation is based on separately analyzing the inner level (CPU, I/O equipment, and memory) and the outer level (terminals, and communication lines). The numerical experiments clarified the characteristics of the system response time.

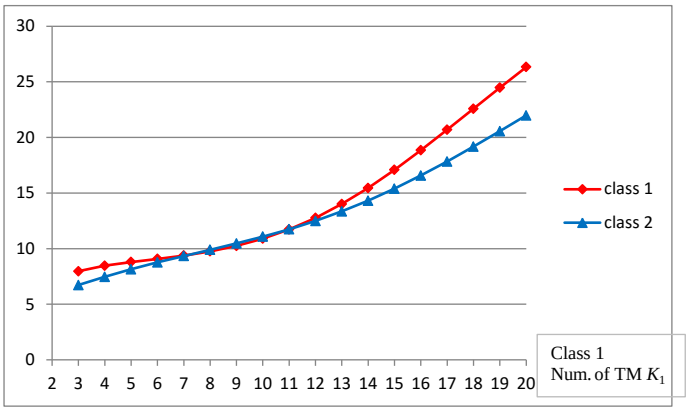
In the future we plan to examine the accuracy of the proposed approximation technique by comparing it with exact solutions or simulation results. Of course since there is the limitation to calculate the product form solution of the inner level because the number of states of the inner level quietly increases when the number of nodes or the number of job classes increase. the comparison of the approximation results and the simulation results is performed within this limitation.

References

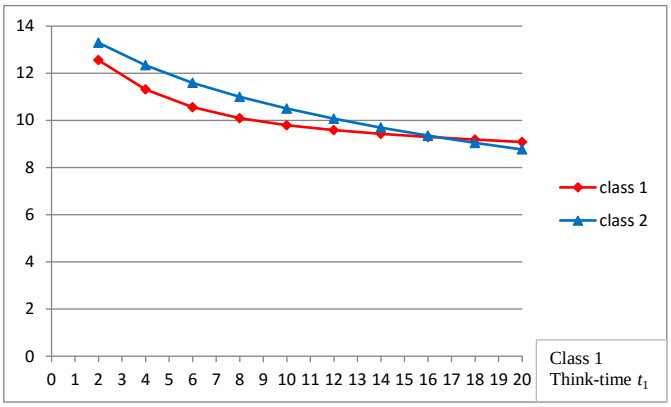
[1] F. Basket, K. M. Chandy, R. R. Muntz, and F. G. Palacious, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *J. ACM*, 22(2):248-260, April 1975.

[2] A. Gandhi, S. Doroudi, M. Harchol-Balter, and A. Scheller-Wolf, "Exact Analysis of the M/M/k/setup Class of Markov Chains via Recursive Renewal Reward," SIGMETRICS'13, pp. 153-166, June 2013.

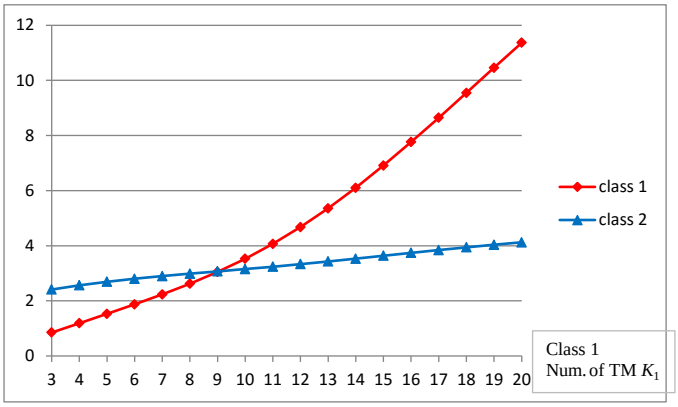
[3] M. Hirose, M. Shiratori, M. Noorafiza, R. Tsuboi, I. Koike, and T. Kinoshita, "Queuing Network Approximation Technique for Evaluating Performance of Computer Systems with Finite Input Source," CSC2015, pp. 9-15, July 2015.



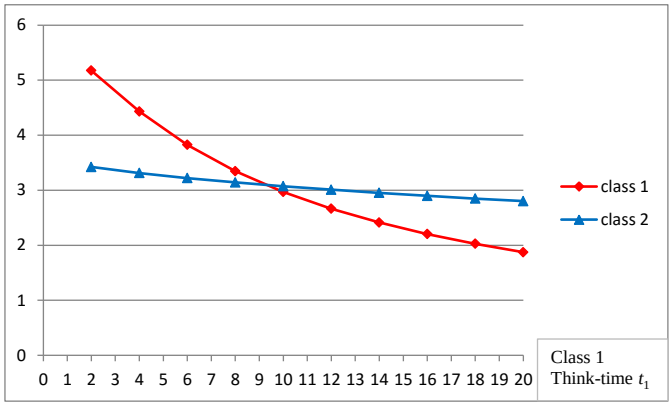
(a) System response time
(K_2=6, (t_1, t_2)=(20, 10))



(c) System response time
((K_1, K_2)=(6, 6); t_2=10)



(b) Mean number of jobs in inner level
(K_2=6, (t_1, t_2)=(20, 10))



(d) Mean number of jobs in inner level
((K_1, K_2)=(6, 6), t_2=10)

Figure 7: Experimental results
(τ_{10}=1.0, τ_{11}=τ_{12}=1.0; τ_{20}=1.0, τ_{21}=τ_{22}=1.0)

[4] T. Kinoshita and Y. Takahashi, "A Queuing Network Modeling and Performance Evaluation Method for Computer Systems with Resource Requirement," *IEICE D-I*, J 82-D-I(6):701-710, June 1999.

[5] T. Kinoshita and X. Gao, "Queuing Network Approximation Technique for Evaluating Performance of Computer Systems with Memory Resources," PDPTA2010, pp. 640-646, July 2010.

[6] H. Kobayashi, *Modeling and Analysis*, Addison-Wesley Publishing Company, Inc. 1978.

[7] T. Kurasugi and I. Kino, "Approximation Method for Two-layer Queueing Models," *Performance Evaluation*, 36-37:55-70, 1999.

[8] O. E. Oguike, M. N. Agu, and S.C. Echezona, "Modeling Variation of Waiting Time of Distributed Memory Heterogeneous Parallel Computer System Using Recursive Models," *International Journal of Soft Computing and Engineering*, 2(6):70-77, Jan 2013.

[9] A. Razali, T. Kinoshita, and A. Tanabe, "Queuing Network Approximation Technique for Evaluating Performance of Computer Systems with Multiple Memory Resource Requirements," PDPTA2012, pp. 758-763, July 2012.

[10] J. A. Rolia and K. C. Sevcik, "The Method of Layers," *IEEE Trans. on Software Engineering*, 21(8):689-700, Aug. 1995.

[11] M. Takaya, M. Ogiwara, N. Matrazali, C. Itaba, I. Koike, and T. Kinoshita, "Queuing Network Approximation Technique for Evaluating Performance of Computer Systems with Memory Resource used by Multiple job types," CSC2014, pp. 41-46, July 2014.

Toshiyuki Kinoshita received his Master of Science from Tokyo University in 1977 and worked for Systems Development Laboratory, Hitachi Ltd. for 28 years. He received his Ph.D. of Science from Tokyo Institute of Technology in 2000 and is a Professor at Tokyo University of Technology since 2005. Kinoshita is majoring in Computer Security, System Programing and Queuing Theory.

Noor Afiza Mat Razali received her Master of Engineering from Tokyo University of Technology. She is a lecturer at National Defense University of Malaysia and currently on study leave for her Ph.D. at Tokyo University of Technology. Prior to becoming lecturer at National Defense University of Malaysia, she has experiences working in various multinational IT companies including Hewlett Packard.

Kaori Katsumata is the 4th year student of Tokyo Institute of Technology. She is majoring in Queuing Theory for Evaluating Computer System Performance.

A Pipelined Implementation of Hash Stream1-Synthetic Initialization Vector Encryption Algorithm

Maththaiya Durai* and Behnam S. Arad†
California State University, Sacramento, CA 95819-6021 USA

Abstract

Data security is a major concern for everyone in today's informational world. Encryption is the process of encoding messages or information in such a way that only authorized parties can access it. It is one of the major information security solutions. *Hash Stream1-Synthetic Initialization Vector* (HS1-SIV) is a recently developed and fast encryption algorithm. We proposed a hardware accelerator for the HS1-SIV encryption algorithm. The implementation relied on parallelism and pipelining to increase message encryption throughput. A unique feature of the proposed pipelined design is that the key generation steps including *CHACHA* stream cipher, *HS1-Hash* and *HS1-pseudo* random function were performed in parallel. This lowered the delay associated with each round of encryption and reduced the overall encryption delay of a plaintext block. This led to an increase in the message encryption throughput. The hardware realization of HS1-SIV encryption algorithm involved designing a hardware data path and control unit, modeling them in System Verilog hardware description language, validating, and synthesizing the design using a 90nm hardware cell library. The proposed design was thoroughly verified using a System Verilog layered test bench. The extent of verification was measured by using System Verilog Functional Coverage. The expected results used in validating the implementation were generated as part of the layered test bench infrastructure. The proposed pipelined model is very efficient and can encrypt messages at the rate of 457 Gigabytes per second.

Key Words: Encryption, hardware accelerator, System Verilog, pipelining, simulation, synthesis.

1 Introduction

Storage, processing, and communication of data in electronic form have significantly increased over the past twenty-five years. Availability of data in electronic form has attracted nefarious elements. Many high profile data hacks in the recent years highlight the need for data security. One approach to achieve data security efficiently is through encryption. It involves encoding messages or information in such a way that only authorized parties can access it.

To ensure security and confidentiality the message is encrypted using an encryption algorithm (cipher). The encrypted message (cipher text) yields the original message only upon decryption. Encryption key (pseudo random number) generated by an algorithm forms the crux of the encryption process. An authorized recipient who has the key can easily decrypt the message while an unauthorized interceptor cannot. Due to its effectiveness, encryption has been long employed by government agencies and military to ensure the secrecy of communication.

Authenticated encryption uses block ciphers to both encrypt and generate a Message Authentication Code (MAC) in order to provide confidentiality and authentication, respectively. HS1-SIV is one of the faster authenticated encryptions. It requires a single key for both encryption and decryption, which is independent of the message and the cipher itself. There are three variations of HS1-SIV algorithm (HS1-SIV-LO, HS1-SIV, and HS1-SIV-HI) depending on the number of bytes used in the hashing part of algorithm, the size of the synthetic IV, the number of internal rounds used by the stream cipher, and the collision level of the hashing algorithm [5].

In this paper, we propose a hardware model for implementing the HS1-SIV algorithm using the System Verilog hardware description language. The model relies on pipelining to greatly improve the throughput of the design. The model was thoroughly verified using a layered test bench. Then, the verified model was synthesized using the Synopsys Design-Compiler tool to get an estimate of the number of gates, area and timing of the hardware model.

The rest of the paper is organized as follows. Section 2 covers an overview of the HS1-SIV encryption algorithm. Section 3 discusses the design and architectural view of the hardware implementation of the HS1-SIV encryption algorithm. Section 4 describes the modeling of the HS1-SIV algorithm using System Verilog. Section 5 covers the verification of the hardware model. Section 6 covers the synthesis of the hardware model using the Synopsys Design Compiler synthesis tool. Finally, Conclusions are stated in Section 7.

* Department of Electrical and Electronic Engineering. Email: maththaiyadurai@csus.edu.
† Department of Computer Science. Email: arad@csus.edu.

2 HS1-SIV Encryption Algorithm

2.1 Overview

This section introduces the concept of HS1-SIV encryption algorithm. HS1-SIV is an authenticated-encryption algorithm developed by Krovetz in 2014. It is designed to exploit 32-bit multiplication and Single Instruction Stream Multiple Data Stream (SIMD) processing, which are well-supported on almost all current CPUs [5].

The HS1-SIV algorithm is a symmetric cipher. In symmetric ciphers, a single secret key is used for both the encryption and decryption, whereas in asymmetric ciphers, there are two sets of keys known as private and public keys. The plaintext is encrypted using the public key and can only be decrypted using the private key [9].

HS1-SIV uses a new pseudo random function called HS1 to provide authenticated encryption via Rogaway and Shrimpton's SIV mode [10]. HS1-SIV maintains full integrity and confidentiality over the message. HS1-SIV is designed to have the features such as competitive speed on multiple architectures, provable security, general-purpose PRF, scalable and nonce misuse resistant [5]. HS1 uses a universal hash function to accept arbitrary strings and a stream cipher to produce its output. SIV, as defined in [10], uses a block-cipher-based PRF to create a synthetic IV (an SIV) from given associated data and plaintext.

"HS1-SIV uses HS1 to instantiate SIV mode. If *A* is the associated data, *M* is the plaintext, and *N* is nonce, then the SIV is defined as the first 16 bytes of HS1(*A*||*M*, *N*) and cipher text *C* is defined as all but the first 16 bytes of HS1(SIV, *N*) XOR'ed with *M*. The SIV and cipher text are bundled together to create the final cipher text. If (*A*, *M*, *N*) is repeated, then an observer knows this fact because the scheme is deterministic and the SIV will be identical, but no security degradation otherwise occurs. Supplying *N* as a nonce thus improves security by masking repeated encryptions.

HS1 operates by pairing an almost-universal hash function with a stream cipher. When given an input (IV) pair, HS1 uses the hash function to hash the input, it then XOR's this hash result with the stream cipher's key and uses the HS1 IV as the stream cipher's IV. The stream cipher produces as many bytes as desired. As long as a (hash result, IV) pair is never repeated, and the stream cipher is secure against related-key attacks, the stream cipher will produce independent pseudorandom output streams. We introduce a new hash HS1-Hash which we use for the almost- universal phase of HS1, and Bernstein's *CHACHA* is used as the stream cipher**"** [5].

2.2 HS1-SIV Process

An authenticated-encryption scheme is a shared-key encryption scheme which provides both privacy and authenticity. The encryption algorithm takes a key, a plaintext, an associated data and a nonce, and returns a cipher text and an SIV. The decryption algorithm takes a key, a cipher text, an SIV, an associated data and a nonce, and returns either a plaintext or an invalid indicator.

k = *HS1-subkeygen [b, t, r, ℓ] (K)*

M' = $A||M$ || $|A|||$ *toStr(8, jMj)*

T = *HS1 [b, t, r] (k,* M' *, N, ℓ)*

C = *M_ HS1 [b, t, r] (k, T, N, 64 + jMj) [64, jMj]*

Inputs: *(K, M, A, N)*

Output: *(T, C)*

Where K, a non-empty string of up to 32 bytes
M, a string shorter than 264 bytes
A, a string shorter than 264 bytes
N, a 12-byte string
(T, C), strings of ℓ and jMj bytes, respectively

Figure 1: HS1-SIV process, based on [5]

"HS1-SIV has three variations depending on the parameters b, t, r, *and* ℓ. Parameter b specifies the number of bytes used in part of hashing algorithm (larger *b* tends to produce higher throughput on longer messages). Parameter *t* selects the collision level of the hashing algorithm (higher *t* produces higher security and lower throughput). Parameter *r* specifies the number of internal rounds used by the stream cipher (higher *r* produces higher security and lower throughput). Parameter ℓ specifies the byte-length of synthetic IV used (higher ℓ improves security and increases cipher text lengths by ℓ bytes). The following table names parameter sets." [5] [11]

Table 1: HS1-SIV Variations

Name	***b***	***t***	***r***	***l***
HS1-siv-lo	64	2	8	8
HS1-siv	64	4	12	16
HS1-siv-hi	64	6	20	32

Next we describe 4 stages of HS1-SIV algorithm.

2.2.1 SubKey Generation. The first stage of HS1-SIV, *Subkeygen* generates the sub keys from the input key of up to 32 bytes. HS1-SIV uses *CHACHA12*, a stream cipher to produce pseudorandom string. This phase produces three keys denoted by *ks*, *kn*, *kp* which are used in the later hash stages.

T= Chacha[r] (K, 0, N, 0^y)
ks= T[0, |C|]
kn= toInts(4, T[|C|, |NH|])
kp= map(mod 2^{60}, toInts(8, T[|C|+ |NH|, |P|]))

Figure 2: *Subkey* generation, based on [5]

where |C|= 32, |NH|= b + 16(t -1), |P|= 8t, y = |C|+ |NH|+ |P|,N = toStr(12, $b2^{48}$ + $t2^{40}$ + $r2^{32}$ + $\ell 2^{16}$ + |K|), toInts(*n*, *S*) is the vector of integers obtained by breaking *S* into *n*-byte chunks and Little-endian interpreting each chunk as an unsigned integer, and | | represents concatenation.

2.2.1.1 CHACHA Stream Cipher. "CHACHA stream cipher takes four inputs, a 32-byte key, an initial counter value, a 12-byte *Iv*, and a plaintext. CHACHA produces the cipher text which is as long as the plaintext. CHACHA, builds a 4×4 matrix, and transforms the matrix through 12 rounds, and adds the result to the original matrix to obtain a 16-word (64-byte) output block. CHACHA, uses 4 additions and 4 exclusive-or's (XOR's) and 4 rotation operations to update 4 32-bit state words. However, CHACHA applies the operations in a different order, and in particular updates each word twice rather than once. CHACHA updates a, b, c, d as follows:

```
a += b;   d ^= a;   d <<<= 16;
c += d;   b ^= c;   b <<<= 12;
a += b;   d ^= a;   d <<<= 8;
c += d;   b ^= c;   b <<<= 7;
```

Figure 3: CHACHA quarter round, Based on [5]

CHACHA first round modifies first, fourth, third, second, first, fourth, third, second along columns, and the second round modifies first, fourth, third, second, first, fourth, third, second along southeast diagonals: The four quarter round words are always in top-to-bottom order in the matrix, to improve diffusion slightly." [1, 2, 7]

```
QUARTERROUND (X0, X4, X8, X12)
QUARTERROUND (X1, X5, X9, X13)
QUARTERROUND (X2, X6, X10, X14)
QUARTERROUND (X3, X7, X11, X15)
QUARTERROUND (X0, X5, X10, X15)
QUARTERROUND (X1, X6, X11, X12)
QUARTERROUND (X2, X7, X8, X13)
QUARTERROUND (X3, X4, X9, X14)
```

Figure 4: CHACHA round order, based on [5]

2.2.2 Message Padding. In this phase, the plaintext (M), associated data (A), length of the plaintext data and length of the associated data are concatenated to produce the plaintext prime. The later stages of hash operate on the produced plaintext prime. Figure 5 summarizes this phase.

$$M' = A \,||\, M \,||\, |A| \,||\, |M|$$

Figure 5: Message padding, based on [5]

2.2.3 SIV Generation. The next stage in HS1-SIV is SIV generation based on *HS1-PRF*. *HS1-pseudo* random function is a composition of *HS1-hash*, an almost universal hash function, with CHACHA, a stream cipher.

$$Ai = HS1\text{-}Hash\,(kn[4i, b/4], kp[i], M) \text{ for each } 0 < i < t$$
$$Y = Chacha[r](pad(32, A0 \,||\, A1 \,..\, ||At\text{-}1) \wedge ks), 0, N, 0^{y})$$

where ks -subkey string of 32 bytes,
kn - a vector of b/4 + 4(t-1) integers,
kp - a vector of t integers,
M - an input string of any length,

Figure 6: SIV Generation, Based on [5]

2.2.3.1 HS1-Hash. First stage in SIV generation is breaking the plaintext prime into a vector of intermediate strings using *HS1-hash*. It is a composition of two hashes, *NH hash* and *Poly hash*. Similar to the techniques used in UMAC and VMAC, the *NH hash* is used to reduce the input by a fixed ratio to an intermediate string which is then hashed to a fixed size by a polynomial evaluation. To reduce the chance of collision, this hashing procedure is repeated *t* times, with different keys, for a higher collision probability [5].

$$ai = (NH(kn, mi) + |Mi| \bmod 16) \bmod 260, \text{for} 1 < i < n$$
$$h = (kp + a1 * kp^{n-1} + a2 * kp^{n-2} + \ldots + a0) \bmod (261 - 1)$$
$$Y = toStr(8, h)$$

where $n = \max(\lceil |M|/b \rceil, 1)$,
$|Mi| = b$ *for each* $1 < i < n$,
$mi = toInts(4, pad(16, Mi))$ *for each* $1 < i < n$,
Y, an 8 byte (if t<4) or 4 byte (if t>4) string

Figure 7: *HS1-Hash*, Based on [5]

2.2.4 HS1-SIV-PRF. The last stage in HS1-SIV applies HS1 pseudo random function to the synthetic IV produced by SIV generation stage and produces the cipher text. In this stage, the SIV produced by the previous stage is used as a plaintext. *HS1_ hash* is used to break the plaintext into vector of intermediate strings and applied to *CHACHA* stream cipher to produce the output string. The output string of *CHACHA* is XOR'ed with the input message and concatenated with synthetic IV to produce the final cipher text.

3 Hardware Realization of HS1-SIV

In this section, we propose a hardware realization of the HS1-SIV algorithm. In the subsequent sections we show that the pipelined design is fully functional and synthesizable. This means that the Register Transfer Level (RTL) description of the designs can be converted to an optimized gate-level net list using a logic synthesis tool. We developed a non-pipelined implementation primarily to assess the efficiency of the pipelined implementation. That implementation is not described here due to space limitations. It is described in detail in [3].

In this work, we considered incoming message and associated data of 512-bits, a key of 256 bits, and a nonce of 96 bits wide. If the message packet is of variable size, then the design requires FIFO logic to store the entire message before the hashing process. We also designed Non-pipelined

design of the algorithm, to compare the results with pipeline design and to estimate the efficiency of the pipeline design. As part of this work, we focused on the pipeline design and the non-pipeline design results were mentioned in the later chapters.

In the following subsections we introduce a pipeline implementation of the HS1-SIV algorithm. The design hierarchy of HS1-SIV pipelined design is shown in Figure 8.

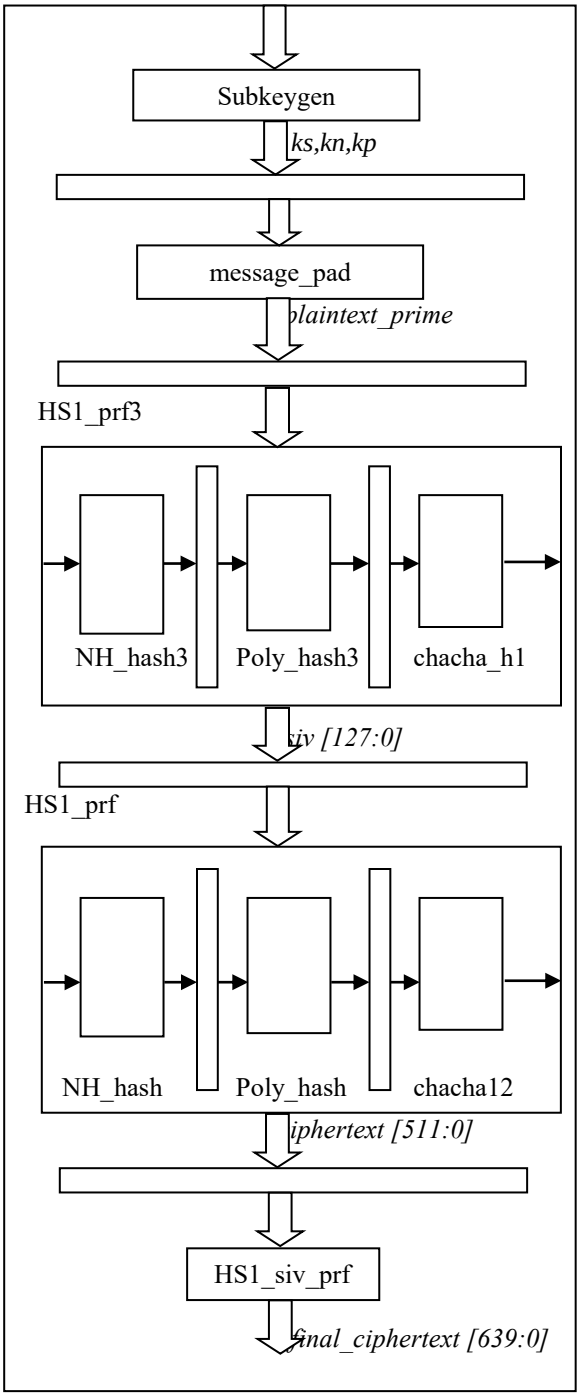


Figure 8: Pipeline HS1-SIV Design

3.1 Subkeygen Stage

The main task of subkeygen stage is to generate sub keys from the input key using *CHACHA12* stage and keys_gen stage. *CHACHA12* stage takes the length of pseudo random string, the key, and the initial counter value as inputs and produces the pseudo random string of as long as needed. Keys_gen stage takes the produces pseudo random string and breaks into three sub-keys.

The CHACHA12 stage produces 512-bit pseudo string as its output. The inputs and outputs of this stage are shown in Figure 9. Each quarter round in the CHACHA12 performs twelve rounds of CHACHA transformations. The quarter round stage takes four 32-bit inputs, rotates the four inputs, and produces four 32-bit outputs. This stage uses 4 additions, 4 XOR's and 4 rotations to update 4 32-bit state words.

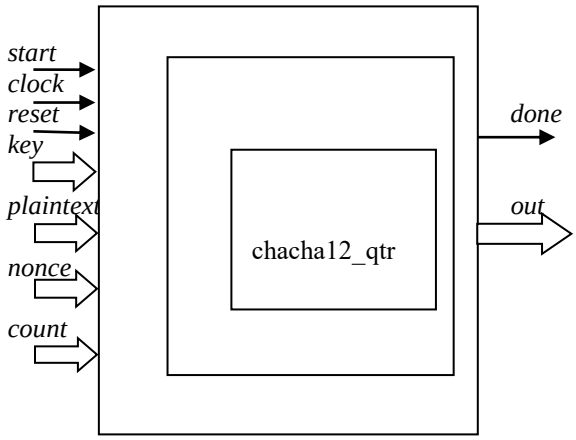


Figure 9: *CHACHA12* Stage

3.2 Message Padding Stage

This stage concatenates the plaintext, the associated data, and two 64-bit parameterized inputs and produces 1536-bit plaintext prime string as output.

3.3 HS1_PRF3 Stage

HS1_prf3, pseudo-random function is a composition of *HS1-hash* and *CHACHA12* stream cipher. *HS1-hash* is itself a composition of two hash functions: NH hash and polynomial evaluation hash. CHACHA12 takes the produced string from HS1-hash function as inputs and produces pseudorandom string as output. The HS1_prf3 stage block diagram is shown in Figure 10.

3.4 HS1_PRF Stage

HS1_prf and HS1_prf3 stages are similar, except for the length of the input plaintext and the output cipher text. The 128-bit cipher text produced by HS1_prf3 stage is synthetic IV (SIV). The SIV is used as the input plaintext for the HS1_prf stage, which produces 512 bits cipher text.

CHACHA12 takes the produced string from *HS1-hash* function as inputs and produces a 512 bits pseudo-random string as output.

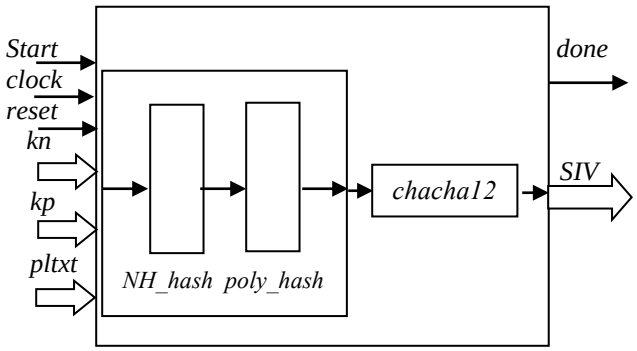


Figure 10: HS1_prf3 stage

3.5 HS1_SIV_PRF Stage

HS1_siv_prf stage concatenates both *SIV* and the cipher text and produces the 640-bit final cipher text. The entire hashing process is performed for each sets of data received by the first stage.

4 Modeling of HS1-SIV

All the blocks in the design hierarchy are modeled in behavioral style of System Verilog. This style models the functionality of a digital circuit at highest level of abstraction. It captures the functionality of a design at an algorithmic level.

The design methodology undertaken in this work is the bottom-up methodology [8]. In this approach, the leaf components in the design hierarchy are developed first and the higher-level components are constructed by instantiating and inter-connecting the subcomponents.

A key aspect of the proposed model is the efficient implementation of multiplication operation. Multipliers are the key components of the design. In this design, Modified Booth algorithm is used for 32-bit multiplication to improve the clock frequency of the design. Modified Booth multiplier reduces the number of iteration step to perform multiplication as compare to conventional steps.

We utilized advance features of System Verilog such as new data types, streaming operators, and the new procedural blocks. Other features that are related to verification are described in detail in Section 5.

System Verilog offers many new data types to be used in a design such as logic, *enum* and *struct*. Logic is a 4-state variable, like the Verilog reg type. It can be declared as a vector. Enum type is an enumerated net or variable with a labeled set of variables, similar to C enum type, but with additional syntax and semantics for modeling hardware. Struct type is a collection of variables that can be referred to individually or collectively, similar to the C struct type [14].

The streaming operators perform packing of bit-stream types into a sequence of bits in a user-specified order. The streaming operator << or >> determines the order in which blocks of data are streamed: >> causes blocks of data to be streamed in left-to-right order, while << causes blocks of data to be streamed in right-to-left order [6].

The specialized *always_comb*, *always_latch*, and *always_ff* procedural blocks indicate the design intent. The software tools do not need to infer the designer's intend. If the content of a specialized procedural block does not match the rules for that type of logic, software tools can issue the warnings [14].

The proposed pipelined HS1-SIV hardware model consists of several different stages to produce the cipher text. Figure 8 illustrates the top-level block diagram of the design. The complete System Verilog code for the pipelined implementation is provided in [3]. Next, we summarize some modeling choices we made in this work.

4.1 Subkeygen Stage

The block in *subkeygen* is modeled as a System Verilog module which instantiates *CHACHA12* and keys_gen module. *CHACHA12* stage requires 12 rounds of *CHACHA* transformation. Each round is modeled as a module. Each quarter round of *CHACHA* transformation is modeled as a *CHACHA12_qtr* module and is instantiated four times within *CHACHA12* round module to produce one round of *CHACHA* operations.

4.2 Message Padding Stage

This stage is used for message padding using concatenation operation. It applies endian conversion operation on the length of the plaintext and the associated data variables. The size of length variables used in this phase is 64-bits. The output obtained after concatenating the strings is a unique plaintext_prime string, which is used in the next hash function to generate synthetic IV namely, *SIV*. The inputs 512-bit plaintext, subkeys *ks, kn, kp* and *nonce* are buffered in this stage.

4.3 HS1_PRF3 Stage

The *HS1_prf3* stage is modeled as a System Verilog module which instantiates NH_hash3, Poly_hash and *CHACHA12_h1* modules. The inputs 512-bit plaintext, subkeys *ks, kn, kp* and 96-bit *nonce* are buffered in this stage.

NH_hash3 operation is modeled as a System Verilog module and instantiated four times. This is due to the fact that for the parameter *t*=4, *NH_hash3* should be iterated four times with the different combination of the subkey, *kn*. In this phase, *NH_hash* is applied to the plaintext_prime string produced by the mesage_pad stage. This stage generates three 64-bit outputs after hashing the input to generate an output with reduced bits. In addition, this stage breaks the message into *b*-bit blocks hashes the message into 128-bit hash output using NH (universal hash family). NH_hash3

consist of three internal stages called *NH_add3*, *Multiply*, and *adder64*. NH_hash3 converts the endianness of the data (big endian to little endian) and includes a simple adder. This *Multiply* module multiplies the two 32-bit data values it receives from endian 64-bit adder module and generates a 64-bit output value. Modified Booth algorithm is used for 32-bit multiplication to improve the clock frequency of the design. Four instances of the multiply module are used for one hashing routine. This *Adder64* module adds two 64-bit inputs provided by the 32-bit multiplier module. It generates a single 64-bit output. Any carry-out from the addition is discarded.

The polynomial hash function is modeled as a System Verilog module and which is instantiated four times. Poly hash module applies polynomial expression on two 64-bits outputs from the adder64 module, *kp*, and the square of subkey *kp*. This modules produces one 64-bit string as output.

4.4 CHACHA12_h1 Stage

CHACHA12_h1 module is functionally similar to *CHACHA12* module, even though the number of input elements and output elements are lesser.

4.5 HS1 PRF Stage

The *HS1_prf* module is functionally similar to *HS1_prf3* module, the variation being the size of the input plaintext and the output cipher text. *HS1_prf* takes the 128-bit *SIV* produced by *HS1_prf3*. The input goes through the *NH_hash*, *poly_hash*, and *CHACHA12_h2 stages*. *HS1_prf* produces 512-bit cipher text as its output. The inputs 512-bit plaintext, subkeys *ks, kn, kp* and 96-bit *nonce* are buffered in this stage.

4.6 HS1_SIV_PRF Stage

This module is the final stage of HS1_siv algorithm. It takes the 128-bit *SIV* produced by *HS1_prf3*, the 512-bit plaintext and 512-bit cipher text produced by *HS1_prf*. This module applies XOR operation on the 512-bit plaintext and the cipher text and concatenates the XOR output and *SIV* and produces 640-bit final cipher text.

5 HS1-SIV Verification

5.1 Overview

This section describes the layered System Verilog test bench developed to verify the functionality of the proposed HS1-SIV design. The simulation was performed using Synopsys® VCS simulation tool. The test bench fully validated the design by constructing random messages, keys and nonce, passing them to the model, and comparing the actual HS1-SIV output to the expected result. A *scoreboard* block was used to generate the expected value for each test vector.

5.2 Layered Test Bench Verification

The layered test bench comprises of multiple layers to take full advantage of code reuse and automation [15]. It consists of *scenario*, *functional*, *command*, and *signal layers*. The lowest layer is the signal layer that connects the test bench to the DUT. It consists of a System Verilog *interface* with a *clocking block* and appropriate module port (*modport*) constructs. The next layer is the *command layer* which contains *driver* and *monitor* components. The driver converts a single command from the functional layer to input signals for the DUT. The monitor converts the DUT outputs to commands for the functional layer. The next layer up is the functional layer. It converts high-level transactions to commands for the command layer. It validates the transactions by generating the expected results using its *scoreboard* block through the *checker*. The scenario layer converts test cases to transactions for the functional layer.

We implemented a layered test bench as part of a verification wrapper. The wrapper included a System Verilog program block, an interface, the DUT, and clock generation blocks needed for validation of the DUT. Next, we describe different features of System Verilog utilized to realize the layered test bench.

5.3 Interface

A System Verilog interface encapsulates the communication between blocks, allowing a smooth refinement from abstract system-level through successive steps down to lower RTL and structural levels of the design.

Interfaces also facilitate design re-use [12]. In this work, we used two interfaces, namely *interface_top* and *interface_dut*. The *interface_top* was used to establish the communication between the *programs* representing the test bench and the HS1_SIV top module. The *interface_dut* was used to set up the communication between the modules in top module. Two modports were used in the *interface_top*, one for the program and one for the DUT. Seven modports were used in the *interface_dut*, one for the each module in top module.

5.4 Threads and IPC

System Verilog constructs such as fork *join_none* and fork *join_any* can be used to dynamically create new threads, in addition to the standard *fork_join*. These threads communicate and synchronize using events, semaphores, mailboxes, and classic @ event control and wait statements. System Verilog enables design of a powerful and flexible test bench environment, as *Object Oriented Programming* (OOP) objects are created and destroyed, they can run in independent threads [15].

In a layered test bench, each transactor (object) gets a transaction from an upstream object, performs some operations, and then passes the transaction to a downstream object. The channel allows its driver and receiver to operate asynchronously. In this work, standard fork/join was used to create threads, and mailboxes were used to provide inter process communication between the threads. A mailbox is just a FIFO, with a source and sink. The source puts a value into the mailbox, and the sink gets values from the mailbox. It can have a maximum size or can be limited.

5.5 Program

In System Verilog, the test bench can be developed as a program block, which supports multiple implicit timing regions for sampling test bench results, scheduling the design events, observing System Verilog assertions, and taking reactive steps in the test bench [4].

The test bench described in this section consists of a single program. It uses the Object Oriented Programming feature of System Verilog to build random test vectors dynamically. The random test vectors were generated using rand system built-in task inside a Class. The randomization method used in the test bench is Constraint Based, which programs the simulator to limit the selection of randomized value to a specific value-set pool.

5.6 The Environment

In this work, a verification environment was properly initialized and synchronized using System Verilog *mailbox*, avoiding race condition between the design and the test bench. The verification environment automates the generation of input stimuli, and reuses existing models and other infrastructure. The test bench (program) verifies the design until functional coverage reaches 100%. The verification procedure involves generating stimuli randomly, passing them to the design through *interface_top* and verifying the correctness of the results obtained. The generator, agent, driver, monitor, checker and scoreboard are all classes, modeled as *transactor*. They are instantiated inside the Environment class. The program block instantiates the environment class.

5.7 Validation

Scoreboard block has the procedures to generate the expected results of the HS1-SIV design. It then forwards the results to the checker class to verify whether the actual results from DUT and expected results from Scoreboard match, and sets up the pass/fail flag accordingly. This verifies the HS1-SIV design output correctness and proper functional working.

5.8 Functional Coverage

Functional coverage is a measure of how an RTL model matches the specification. In a System Verilog test bench, the coverage is calculated by sampling the values of variables and expressions. These sample locations are known as *cover points*. Multiple cover points that are sampled at the same time, such as when a transaction completes, are placed together in a *cover group*. A cover group is similar to a class, it contains cover points, options, formal arguments, and an optional trigger. Sample function was used to trigger the cover group [15].

A cover group encapsulates all the cover points for the randomized plaintext, the associated data, the key and the nonce for gathering coverage. These randomized signals defined as the cover points are necessary to verify HS1-SIV design by measuring the functional coverage. We defined our cover points based on the plaintext, the associated data, and the keys to measure functional coverage of our verification. We were able to achieve 100% functional coverage which reflects the thoroughness of the verification process and robustness of the model.

6 Logic Synthesis

In this section, the synthesis of the HS1-SIV hardware model is described. One of the objectives of this work was to develop a synthesizable model of the HS1-SIV algorithm. Logic synthesis is the process of converting a high-level description of a design into an optimized gate-level representation, given a standard cell library and certain design constraints. The Synopsys® Design Compiler was the synthesis tool used in this work.

Logic synthesis tools take the HDL model of a design, information on cells from a technology library, and the design constraints. A technology library can have simple cells, such as basic logic gates like AND, OR, and NOR, or macro cells, such as adders, multiplexers, and special flip-flops. The synthesis tool converts the HDL design into an optimized gate-level net list satisfying the given design constraints such as timing, area, testability, and power using the cells from the technology library [8].

The proposed HS1-SIV model was synthesized towards a 90 nm technology library. We synthesized both the pipelined and the non-pipelined implementations. The timing results showed that the pipelined design can operate at 71MHz. This was based on the synthesis tool being configured for the high synthesis effort. We expect the design to be able to operate at a higher frequency if it is synthesized using a more recent technology library.

The *HS1_hash* function for the proposed HS1-SIV design with 512-bit block size, take 30 clock cycles to complete. This is the bulk of the processing for a short message. There are number of multiplication operations in the hash function used. The Modified Booth Algorithm is used for faster multiplication. It can improve speed but takes up more hardware area. Each message block uses 114 clock cycles to become cipher text. Further parallel processing can improve this speed.

The synthesis results for HS1_SIV are tabulated in Table 2. It shows the clock frequency, the timing slack, and the cell

area generated after synthesis.

Table 2: Synthesis results

Design	Clock period	Timing slack	Area report
Non-pipeline	85ns	0.3 ns	452481.50 um^2
Pipeline	14ns	0.13 ns	1200432.50 um^2

Speed up of the pipeline design is the ratio of total time taken by the non-pipeline design to encrypt *N* 512-bit messages to that of the pipeline design:

$$\text{Speed up} = 20*85*N / (114+N) * 14, \text{ where}$$

85ns is the non-pipeline design clock period, 20 is the number of steps in the non-pipelined design, 14ns is the clock period the pipeline design, and 114 stages is the number of its stages. This leads to speed up of 121 for large messages. More importantly, the pipeline design can encrypt a 512-bit message per cycle after the first 114 cycles where the pipelined is filled. It can encrypt messages at a very efficient the rate of 457 Gigabytes per second.

7 Conclusion

We proposed an efficient pipelined implementation of a recently developed fast encryption algorithm called HS1-SIV. Our synthesis results showed that the pipelined design can achieve encryption bandwidth of 457 Gigabytes per second. This high encryption rate is achieved by breaking each round of the encryption process into several stages. This hardware concurrency increases the message encryption throughput and makes the hardware model suitable for time-critical encryption applications.

A layered test bench in System Verilog hardware description and verification language, was utilized to thoroughly validate the pipeline design. The test bench included *Functional Coverage* to assess the verification progress of the design features ensuring the design is fully validated. Several unique features of the language including *Interface, Program*, and also *OOP* concept were also used to validate the design. The model was synthesized using Synopsys® Design Compiler tool based on a 90nm technology library.

Acknowledgement

In this work we utilized EDA tools set donated to California State University, Sacramento by Synopsys® Inc.

References

[1] Daniel J. Bernstein, "Chacha, a Variant of Salsa20," Workshop Record of SASC, 8:3-5, 2008.

[2] Daniel J. Bernstein, "The Salsa20 Family of Stream Ciphers," *New Stream Cipher Designs*, Springer, Chapter 8, pp. 84-97, 2008.

[3] Maththaiya Durai, *A Hardware Implementation of HS1-SIV Encryption Algorithm*, Master's Project, California State University, Sacramento, 2015.

[4] Gopi Krishna and Naresh Maddipati, Program Block, http://www.testbench.in/SV_24_PROGRAM_BLOCK.html, January 2007.

[5] Ted Krovetz, HS1-SIV (Draft v2), *CAESER Submission*, 2014.

[6] Bryan Murdock, "System Verilog Streaming Operator," Springer Knowing Right from Left, HYPERLINK http://bryan-murdock.blogspot.com/2014/10/system verilog-streaming-operator.html, http://bryan-murdock.blogspot.com/ 2014/10/systemverilog-streaming-operator.html, October 2014.

[7] Yoav Nir and Adam Langley, "CHACHA20 and Poly1305 for IETF Protocols," HYPERLINK http://datatracker.ietf.org/doc/draft-nir-cfrg-CHACHA20-poly1305, http://datatracker.ietf.org/doc/draft-nir-cfrg-CHACHA20-poly1305, February 2015.

[8] Samir Palnitkar, *Verilog HDL, A Guide to Digital Design and Synthesis*, Prentice Hall, 2nd edition, 2003.

[9] Francisco Rodriguez-Henriquez, N. A Saqib, Arturo Díaz Pérez, and Cetin Kaya Koc, *Cryptographic Algorithms on Reconfigurable Hardware*. Springer, 2007.

[10] Phillip Rogaway and Mihir Bellare, "The Exact Security of Digital Signatures – How to Sign with RSA and Rabin," *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques Saragossa*, Saragossa, Spain, pp. 399-416, May 12-16, 1996.

[11] Phillip Rogaway and Thomas Shrimpton, "Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem," *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, St. Petersburg, Russia, pp. 373-390, May 28-June 1, 2006.

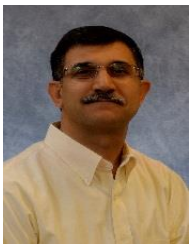
[12] Abhishek Shetty and Dr. Hamid Mahmoodi, "System Verilog Testbench Tutorial," http://userwww.sfsu.edu/necrc/files/synopsys%20tutorials/System_Verilog_Tutorial.pdf, August 2011.

[13] Chris Spear, *SystemVerilog for Verification*, Springer, 3rd edition, 2008.

[14] Stuart Sutherland, *System Verilog for Design*, Springer, 2nd edition, 2006.

[15] Synopsys, *System Verilog Testbench Assistance*, HYPERLINK http://www.synopsys.com/Services/Pages/ SystemVerilogTestbenchAssistance.aspx, http://www.synopsys.com/Services/Pages/SystemVerilogTestbenchAssistance.aspx.

Maththaiya Durai is a Graphics Hardware Engineer for a major chip manufacturing company in the USA. He earned his Bachelor of Engineering Degree in Electronics and Communication Engineering from Coimbatore Institute of Technology, Coimbatore, India, in 2012. He received his Master of Science degree in Electrical and Electronic Engineering from California State University, Sacramento, USA, in 2015.



Behnam Arad is a Professor of Computer Science at California State University, Sacramento. His research interests include low-power SoC Design and hardware security. He earned his PhD degree in Electrical Engineering with a minor in Computer Science from Louisiana State University in 1997.

Consolidation of Data in Multiple Databases

Kasi Periyasamy and Karamveer Yadav*
University of Wisconsin-La Crosse, La Crosse, WI 54601, USA

Abstract

When multiple databases that use same or similar data are used in an application domain for efficiency, ease of access and so on, inconsistencies may arise in the databases if they are not synchronously updated. Even if they are updated synchronously, data migration or data exchange processes between the databases may create problems because of the differences in naming of attributes, data format and constraints imposed on these attributes. This paper focuses on consolidating multiple databases into a single database by focusing on each attribute in the databases. The proposed method consists of three phases - linguistic matching that focuses on names of entities in the databases, structural matching that deals with the structures of schemas and subschemas in the databases and finally constraint matching which elaborately checks the constraints imposed on data elements in each database. The outcome of the process is to create one consolidated database and to let the user interactively select the elements from each database including the data. The paper describes the method and also briefly addresses the complexity of constraint matching phase.

Key Words: Schema matching; constraint matching; content matching.

1 Introduction

Data management is one of the important problems due to unprecedented increase in the amount of data being created and maintained in a broad range of application specific databases. It is quite common to use multiple databases for maintaining similar data, possibly with subtle differences, to improve access or efficiency of data processing. As an example, a university may have student information stored in multiple databases, one for storing the students' academic information (possibly by the Registrar's office) and another one for storing financial aid information (possibly by the Financial Aids office). Since the two databases are used by two different units within the university, it will be convenient for them to create and maintain their own databases for security, ease of access, efficiency and so on. The major problem in this scenario is the duplication of information in these databases which may lead to inconsistency if they are not synchronously updated. Consider, for example, an update on a student's GPA at the end of a semester. Most likely, the academic database maintained by the Registrar's office will be updated with this information because it is in their domain of activities. However, if the same information is not updated in the financial aid database, the student may be denied financial aid for the next semester. Such an inconsistency is not easy to find because usually there will be no direct link between these two databases. One possible solution for this problem is to create a common database for both applications (academic and financial aid) but it will destroy the very purpose of creating multiple databases in the first place for the sake of ease of operation, efficiency and so on. An alternate solution would be to find the commonalities and differences between the databases and create a common database only for those data elements that are shared between the two databases.

The focus of this paper is to find out the commonalities and differences of data elements between the two databases and suggest what can be included in the common database so that both applications can equally use the common database. The paper will not address issues concerning creation and maintenance of additional, individual databases by each application and synchronization between the common and individual databases. Such issues can be dealt with separately. The core of the paper consists of two parts - *schema matching* and *constraint matching*[1]. In the first part, the paper discusses how the names and data types of attributes, and structures of schemas in two databases are matched and the second part deals with matching constraints involved in the data types of attributes in the two databases. Together, the process helps identifying common elements between the two databases selected interactively by the user.

A brief discussion on related work is given in the next subsection. Section 2 starts with a brief introduction on the structure of schemas and a short description of the knowledge base used in the matching process. This is followed by Section 2.2 that describes linguistic matching of attributes in two schemas. The discussion then continues by structural matching of the same two schemas as described in section 2.3. The results of linguistic matching are used in the structural matching process. Section 2.4 describes constraint matching approaches

*Department of Computer Science, Email: {kperiyasamy, yadav.karamve}@uwlax.edu

[1]While many related works in this area describe *schema matching* that also includes *constraint matching*, the authors believe that *schema matching* focuses on the database schema apart from the data created using these schemas. Therefore, the authors use *schema matching* to refer to those elements described by a schema such as name and data type, while *constraint matching* is used to refer to the constraints imposed on the data values specifically to identify the range of data values impacted by these constraints.

used by the authors. Finally, the paper concludes with the discussion on the importance of all three steps - linguistic matching, structural matching and constraint matching for finding commonalities between databases. Continuing work by the authors in this direction also are described in the same section.

1.1 Related Work

A pioneer work on schema matching was done by Madhavan and others [7] who developed a schema matching algorithm called *Cupid*. This algorithm was sufficiently general to be applicable to a variety of applications. As stated in [7], the classification of schema matching techniques includes three major types - *linguistic matching*, *structural matching* and *instance matching*. Ten years later, the same authors published another detailed classification of schema matching algorithms that can be considered as a good source of reference in this area [2]. The recent classification includes some discussions on *schema mapping*, a high-level relationship between a source schema and a target schema [3]. Schema mapping will be quite useful in deciding what data values from one schema can be exchanged with another schema. Automation of schema matching process has been discussed in [10, 12]. A survey of ontology-based schema matching techniques can be found in [4]. Melnik and others [8] have used graph matching algorithms for matching structures of schemas. Work presented in this paper also uses graphs for structural matching and also includes a knowledge base in the matching process. A preliminary work on schema matching by one of the authors is presented in [9].

Instance matching, an extension of schema matching, describes how the actual data in two tables are compared; this process will be utilized in data exchange and data migration applications. As part of *instance matching*, the constraints imposed on data values should also be compared. Valchey and his colleagues has explored *constraint matching* for automation of taxonomies. They described a matching process between objects in an object-oriented system by exploiting dissimilarity measures for collections of objects [14]. More recent work on *constraint matching* is described in detail in [10]. Most approaches discussed in this paper explored data types and multiplicity comparison for set of values by considering the cardinality of attributes as a constraint. Euzenat and Valchey [5] used ontology-based alignment technique which helps in finding semantic interoperability between data sets. Our work on constraint matching presented in this paper is based on simple data type constraints imposed on boundary values. A preliminary work on constraint matching by the authors is presented in [6].

2 The Matching Process

Since *schema* is the underlying structure that is mainly used in the matching process, the discussion begins with the definition of a schema as used in our approach. A *schema* represents a structured representation of data. It consists of a set of attributes and may include other schemas. For the purposes of this paper, we require that a schema should contain at least one attribute. An attribute is the lowest element in a schema that describes a data item. This description includes the name of the attribute, its data type, maximum length of characters of the data item represented by this attribute, whether or not a data item is editable and/or null-able, and finally its default value, if applicable. As far schema matching is concerned, the name and data type of an attribute are the most important factors. A sample schema and its structure is shown in Figure 1. We define schema matching in two phases: *linguistic matching* and *structural matching*. The entire schema matching process is supported by a knowledge base which also includes several dictionaries. The purpose of the knowledge base is two-fold: (i) to maintain several dictionaries in order to eliminate non-application-domain words and common words (e.g., 'and' and 'but') during the matching process, and (ii) to store and reuse some of the previously matched entries.

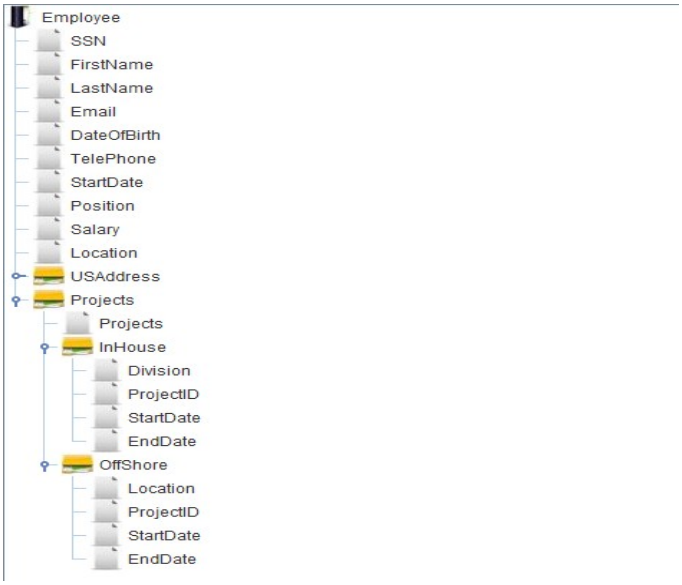


Figure 1: Tree view of a schema

2.1 The Knowledge Base Used in Schema Matching

Our approach uses a knowledge base that includes several dictionaries which store punctuation symbols, special symbols, prepositions, conjunctions, articles, synonyms, hypernyms, acronyms, and abbreviations. Punctuations, special symbols, articles, prepositions and conjunctions are used in eliminating non-domain specific words from the names of entries in the schema. While most of these dictionaries are already preloaded at the time of starting the matching process, the dictionaries corresponding to synonyms, hypernyms, acronyms and abbreviations are updated during the matching process. This is because entries in these dictionaries are application specific. The knowledge base also includes another dictionary called

Name to Concept which is used when two entries belonging to the same concept are named differently. For example, *salary* and *amount* both belong to the concept *money*. So when matching *salary* in one schema with *amount* in another schema, a direct linguistic matching will result in a poor score. This will be enhanced using concept matching.

In addition to storing various dictionaries, the knowledge base also includes a cache storage which stores previously matched entries during a matching session. Once the session is over, this cache is cleared. The purpose of the cache is to speed up the matching process by utilizing previously matched entries. A user may also keep these entries permanently in the knowledge base for future use. This is done by storing the matched entries in a dictionary called *substructures*. The entries in the substructures may be manually tweaked from time to time to make the matching process more efficient.

Figure 2 shows the flow diagram for schema matching.

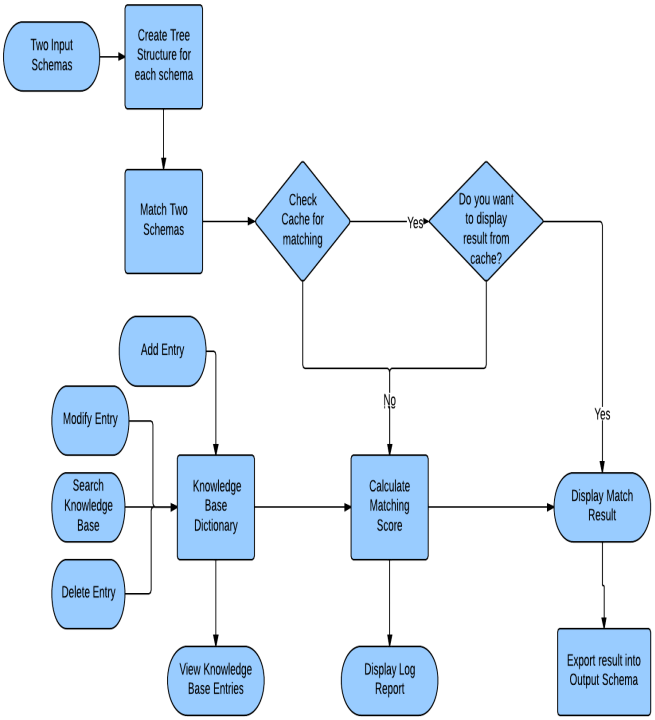


Figure 2: Schema matching process flow diagram

2.2 Linguistic Matching

Linguistic matching is used to match both attributes and schemas. While matching attributes, their names and data types are used. For schemas, only the names of the schemas are used. For the rest of the discussion in this section, let us assume we are matching an entity e_1 with an entity e_2. The linguistic matching process includes three steps - *normalization*, *categorization* and *comparison*. During normalization, the words in e_1 and e_2 are tokenized into spaces, punctuation and special symbols. The purpose of normalization is to eliminate some of the unrelated words such as non-domain words and common words that are found in a built-in dictionary inside the knowledge base. The second step deals with categorizing the normalized words into different groups based on their data types. This step helps identifying the roles of the entities in their respective schemas. Finally, in the third step, the two entities are compared for exact or partial match and a score is derived accordingly. Two entities e_1 and e_2 are said to match exactly if and only if every parameter in the description of one entry matches exactly with the corresponding parameter in the description of the other entry. If they do not match exactly, they are said to match partially. As an example for exact match, consider an attribute named *Student ID* in one schema that is defined as a *String* with an exact size of nine characters. If an attribute from another schema is matched with this attribute, for an exact match, the other attribute must also be named *Student ID* and must have been defined using the data type *String* with exact size of nine characters. If the size restriction is different or the name of the attribute is different (e.g., Stud ID), then there may be a partial match. In some sense, exact matching of attributes during linguistic matching identifies common elements that are named and used in the same way in both schemas and hence in both applications. Figure 3 pictorially describes the linguistic matching process with the support of knowledge base.

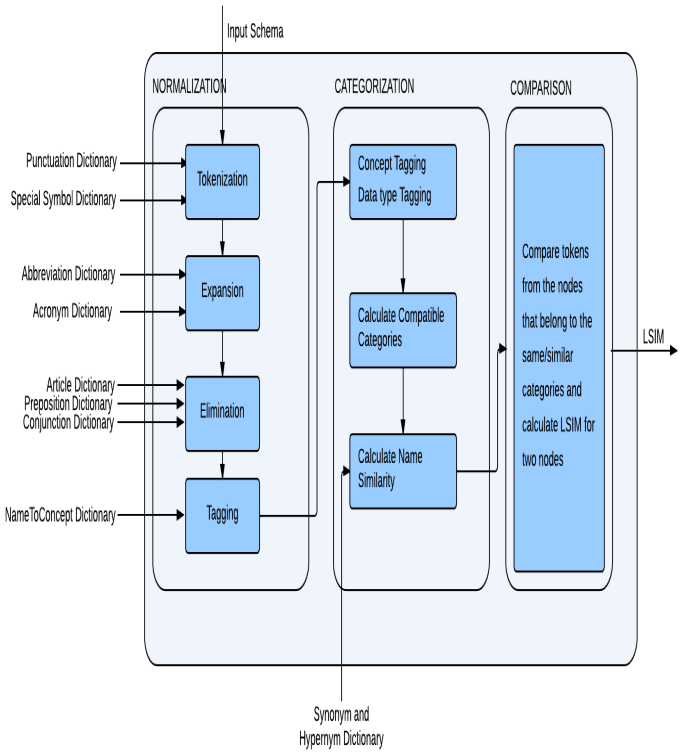


Figure 3: Linguistic matching of two entities

To illustrate the complete process of linguistic matching, consider the entries *DateOfBr* and *Origin*; let *DateOfBr* belong to one input schema and *Origin* belong to another. Both

represent the date of birth of a person. During normalization, the words in *DateOfBr* are tokenized into *Date*, *Of* and *Br*. Among these, *Of* is eliminated because it is one of the common words that appear in a normal dictionary. The word *Br* is expanded using an abbreviation dictionary in the knowledge base. The result is a pair of words *Date* and *Birth*. There is no normalization processing needed for *Origin* and so it is ready for comparison. While matching the two words, the *Name to Concept* dictionary is used because both of these words belong to the same concept, namely *Descent*. This leads to a higher matching score even though both words are literally different. It should be noticed that the matching score of the linguistic matching phase depends not only on the names, but also on the data types of the entities, where the entities are attributes. At the end of the linguistic matching phase, the two entities and their matching score are temporarily stored for later use.

2.3 Structural Matching

Structural matching refers to matching of hierarchical structures of schemas or subschemas. Two schemas exactly (structurally) match with each other if they have the same number of attributes and sub-structures, and the sub-structures also match exactly. The relative positions of the attributes and the sub-structures are not important for an exact match. We use graph theoretic approach to analyze the structures. Since linguistic matching is done separately, structural matching only focuses on syntactic structure of the trees representing the schemas.

Two schemas that do not have an exact match are said to have a partial match. Score for a partial match of two schemas is calculated by computing the cumulative score of their components. For example, consider the scenario of matching two subschemas named *Department* from schema *AcademicScholarship* and *Division* from the schema *AthleticScholarship* as shown in figure 4. Since the two sub-structures are structurally different, there is no exact match. Figure 4 shows the case of matching *DepartmentNo* with *DivisionNumber* and matching *DepartmentName* with *DivisionName*. The scores of these individual matching are recorded. Further, the sub-structure *Major* in the schema *Department* is matched with the sub-structure *Unit* in the schema *Division* which is also recorded separately. The readers are reminded that linguistic matching results are used here to match the components. In this example, the scores of matching the attributes and the sub-structures results in a cumulative score of 0.793 for the two sub-structures. This process can be used to match the entire structure of the two schemas. As in linguistic matching, the outcome of structural matching is also recorded and will be used later to compute the final matching score of the two schemas.

2.4 Constraint matching

As stated in an earlier section, constraint matching is part of instance matching and hence uses both schemas as well as actual

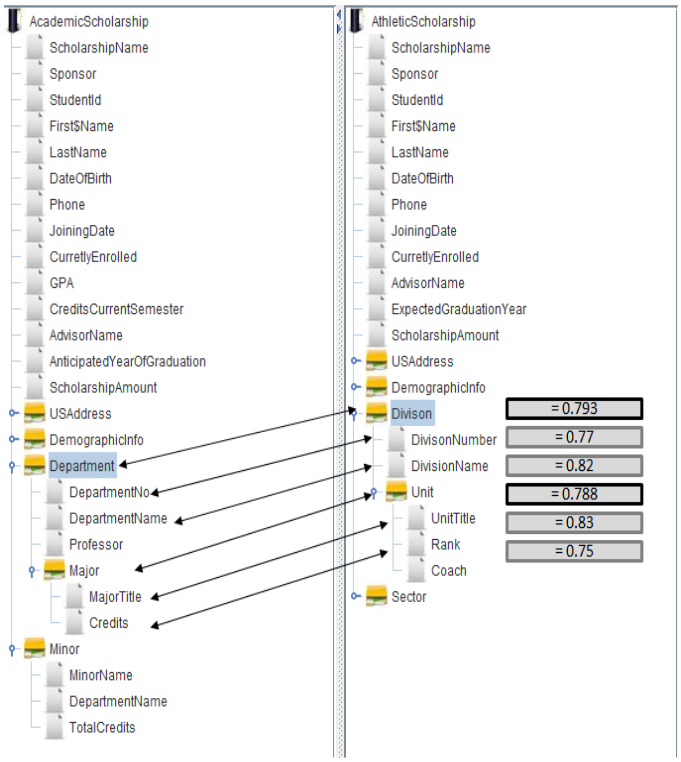


Figure 4: Partial matching of schemas

data from the two databases that are considered. The constraint matching phase strictly follows the previous two phases. Therefore, it is required that the schemas being matched are first considered for linguistic and structural matching. Only those schemas that have satisfactory matching scores in the first two phases[2] will be taken into consideration for constraint matching. Our current implementation on constraint matching considers only six different types of application specific constraints - Less Than (LT), Less Than or Equal to (LE), Greater Than (GT), Greater Than or Equal to (GE), Equal to (EQ) and Not Equal to (NE). Since this phase involves mainly the data associated with the entities, only attributes are considered in constraint matching. The representation of a constraint for an attribute looks like the following:

TableName="Bill"
Attribute="Minimum Payable"
DataType="int"
GreaterThan="99.00"
LessThanOrEqualTo="1000.00"

All constraints are in conjunctive normal form and hence there is an implicit conjunction operator between the constraints. Like the other two phases, the outcome of constraint matching between two attributes is a matching score which can be

[2]A satisfactory matching score is determined by the user by setting a threshold value for an application. It can be predefined or dynamically chosen for matching of each pair of entities during the matching process.

checked against a user-defined threshold. If it is satisfactory, the user can select one of the two attributes and its data for inclusion in the final consolidated database.

There are two steps in the constraint matching process - validation of input constraints and matching of constraints. A flow diagram for constraint matching process is shown in Figure 5. Before matching, it is required to validate each

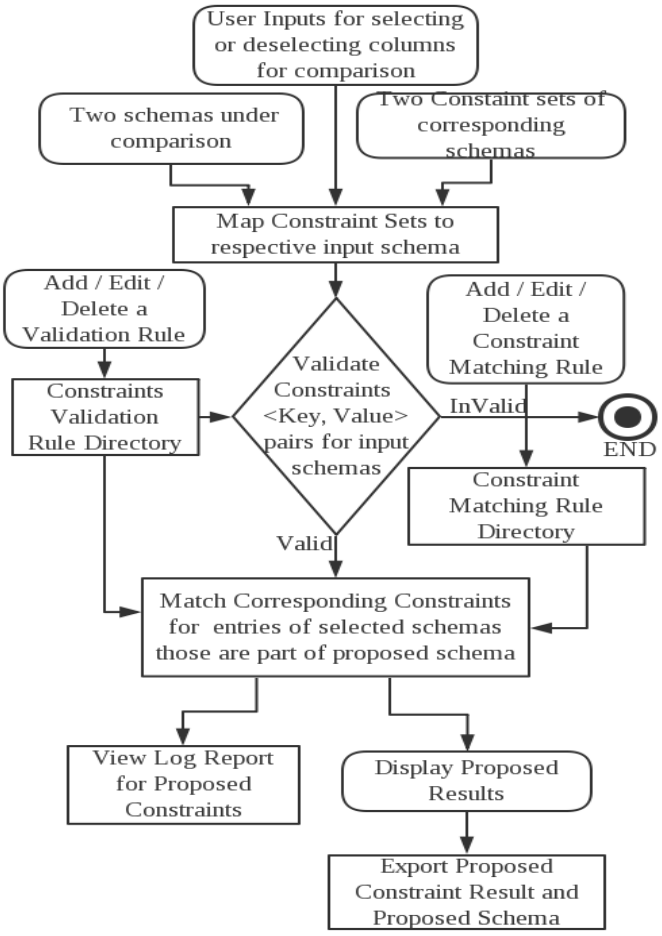


Figure 5: Constraint matching process flow diagram

constraint for the attributes being matched. For example, if the attribute *Minimum Payable* has two constraints that assert that the attribute should be less than or equal to 1000.00 and greater than 100.00, the constraints are valid. However, if the constraints assert that the attribute should be less than or equal to 100.00 and greater than or equal to 500.00, then there is a contradiction because no value of this attribute satisfies these constraints. Figure 6 shows the possible combination of the six operators for validation. Notice that for a given attribute, there could be one to three constraints. For instance, in the previous example, the attribute *Minimum payable* may have three constraints, along with the other two, there can be an additional constraint but its value cannot be equal to 750.00 (for some bizarre reasons). In short, an attribute may have

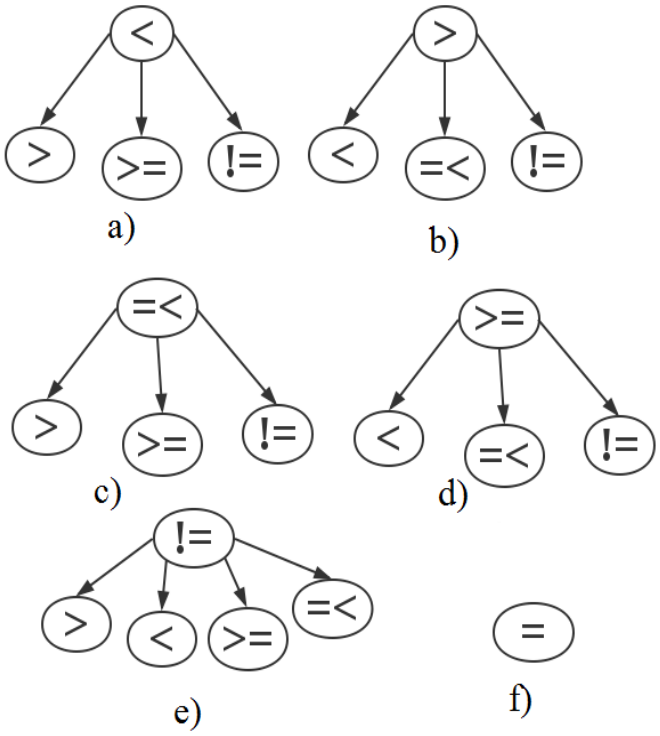


Figure 6: Possible combinations of valid constraints

one or three constraints and their possible combinations are enumerated in Table 1. To support the validation process, we

Table 1: Constraint sets on an attribute

S.No	**Feasible Constraint Set**
1	LT
2	LT and GT
3	LT and GT and NE
4	LT and GE
5	LT and GE and NE
6	LE
7	LE and GT
8	LE and GT and NE
9	LE and GE
10	LE and GE and NE
11	GT
12	GE
13	NE
14	LT and NE
15	LE and NE
16	GT and NE
17	GE and NE

have implemented validation rules that uses the combinations as shown in Figure 6. These validation rules are enumerated in Table 2. In this table, 'V' refers to the current stored value for the attribute in the database table.

Table 2: Validation rules

Constraint in the schema	Validation rule
LT="V1" GT="V2"	V < V1 and V > V2 and V1 > V2
LT="V1" GE="V2"	V < V1 and V ≥ V2 and V1 > V2
LT="V1" NE="V2"	V < V1 and V ≠ V2 and V1 > V2
LE="V1" GT="V2"	V ≤ V1 and V > V2 and V1 > V2
LE="V1" GE="V2"	V ≤ V1 and V ≥ V2 and V1 > V2
LE="V1" NE="V2"	V ≤ V1 and V ≠ V2 and V1 > V2
GT="V1" NE="V2"	V > V1 and V ≠ V2 and V2 > V1
GE="V1" NE="V2"	V ≥ V1 and V ≠ V2 and V2 > V1

Once the constraints are validated for each attribute under consideration, the next step is to match attributes of one table with those in the other table. Notice that these attributes have already been matched for linguistic and structural matching and have been identified as potential candidates for inclusion in the final consolidated database. The matching of these attributes are based on their values and constraints imposed on these values. Given that each attribute may have one to three constraints, there are six possible combinations in matching their values. These are enumerated as 1 ↔ 1, 1 ↔ 2, 1 ↔ 3, 2 ↔ 2, 2 ↔ 3 and 3 ↔ 3. Under each possible combination, there are several matching rules. Due to space constraints in the paper, a subset of these matching rules for 1 ↔ 1 combination that involves "Less Than (LT)" as one constraint is given in Table 3. A similar table for 1 ↔ 1 combination that involves "Greater Than (GT)" as one constraint is given in Table 4. A complete table for 1 ↔ 1 combination is given in [6].

The last columns in Tables 3 and 4 deserve particular attention. This column indicates the suggestion provided by our implementation when matching occurs. The motivation of this consideration is to provide the user with a range of values that satisfy the constraints imposed on both attributes. Notice that these attributes are selected from two different tables and the user is trying to consolidate two of these attributes into one attribute which will become part of a new table. Some of these suggestions are labeled as "No Suggestion" meaning that the user has the freedom to retain the constraint set from any one of the two attributes. To illustrate this process in detail, consider an attribute having a constraint LT="x" and the other attribute having a constraint GT="y" under the scenario "x > y". The range of usable values impacted by these constraints is pictorially described in Figure 7. This particular scenario shows that the new attribute will have both LT="x" and GT="y" as its constraint because the overlapping region in Figure 7 has the only range values that are satisfied by both the constraints. If the scenario becomes "x < y", then there will not be any overlapping region and hence no suggestion is provided to the user (as seen in row 2 in Table 3).

The situation becomes more complex when 2 ↔ 2 and 3 ↔ 3 combinations are considered. For illustration, a 2 ↔ 2 combination is shown in Figure 8 with the constraints LT="x1" and GT="y1" on one attribute, LT="x2" and GT="y2" on the other attribute and the scenario is x1 > x2 and y1 > y2. There are four different regions for this situation as shown in Figure 8. Depending on narrow perspective (regions C and D) or wider perspective (regions A and B), the user may choose appropriate constraints for the new attribute. For example, if the user chooses region C, then the constraint for the new attribute will be LT="x1" and GT="x2" with the scenario x1 > x2. Similarly, if the user chooses region A, the constraint becomes LT="x2" and GT="y1" with the scenario x2 > y1.

Table 3: Constraint Matching Rules for 1 ↔ 1 combination - LT constraint

S.No	First Attribute Constraint	Second Attribute Constraint	Predicate for value comparison	Suggestion
1	LT="x"	GT="y"	x = y	No suggestion
2	LT="x"	GT="y"	x < y	No suggestion
3	LT="x"	GT="y"	x > y	LT="x" and GT="y"
4	LT="x"	GE="y"	x = y	No suggestion
5	LT="x"	GE="y"	x < y	No suggestion
6	LT="x"	GE="y"	x > y	LT="x" and GT="y"
7	LT="x"	NE="y"	x = y	LT="x"
8	LT="x"	NE="y"	x < y	LT="x"
9	LT="x"	NE="y"	x > y	LT="x" and NE="y"
10	LT="x"	EQ="y"	x = y	No suggestion
11	LT="x"	EQ="y"	x < y	No suggestion
12	LT="x"	EQ="y"	x > y	EQ="y"
13	LT="x"	LT="y"	x = y	LT="x"
14	LT="x"	LT="y"	x < y	LT="x"
15	LT="x"	LT="y"	x > y	LT="y"
16	LT="x"	LE="y"	x = y	LT="x"
17	LT="x"	LE="y"	x < y	LT="x"
18	LT="x"	LE="y"	x > y	LE="y"

2.4.1 Complexity of Constraint Matching Process

As evident from the above discussion, the complexity of matching constraints increases enormously with the number of constraints imposed on each attribute. With the increasing complexity, the suggestion provided by the implementation also becomes harder because there will be several isolated ranges of values for the new attribute that are satisfied by the individual constraint sets of both the previous attributes. It is also worth mentioning that our current implementation attempted only three constraints on each attribute and are limited to the basic comparison operators such as "Less Than (LT)" and "Greater Than (GT)". A lot of work needs to be done on complexity

Table 4: Constraint Matching Rules for 1 ↔ 1 combination - GT constraint

S.No	First Attribute Constraint	Second Attribute Constraint	Predicate for value comparison	Suggestion
34	GT="x"	GT="Y"	x = y	GT="x"
35	GT="x"	GT="Y"	x < y	GT="y"
36	GT="x"	GT="Y"	x > y	GT="x"
37	GT="x"	GE="Y"	x = y	GT="x"
38	GT="x"	GE="Y"	x < y	GE="y"
39	GT="x"	GE="Y"	x > y	GT="x"
40	GT="x"	NE="Y"	x = y	GT="x"
41	GT="x"	NE="Y"	x < y	GT="x" and NE="Y"
42	GT="x"	NE="Y"	x > y	GT="X"
43	GT="x"	EQ="Y"	x = y	No suggestion
44	GT="x"	EQ="Y"	x < y	EQ="Y"
45	GT="x"	EQ="Y"	x > y	No suggestion

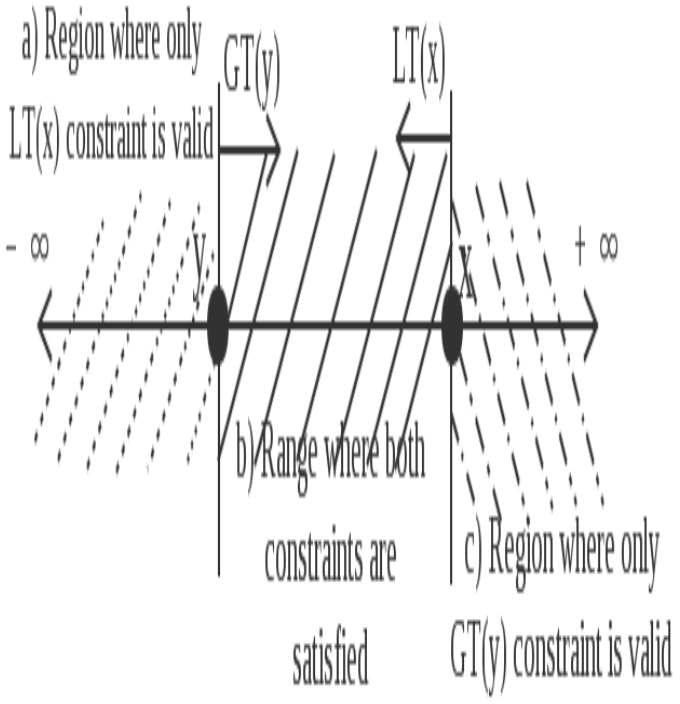


Figure 7: Range of values impacted by LT and GT constraints in 1 ↔ 1 combination

analysis if the application introduces additional constraints apart from these basic comparison operators.

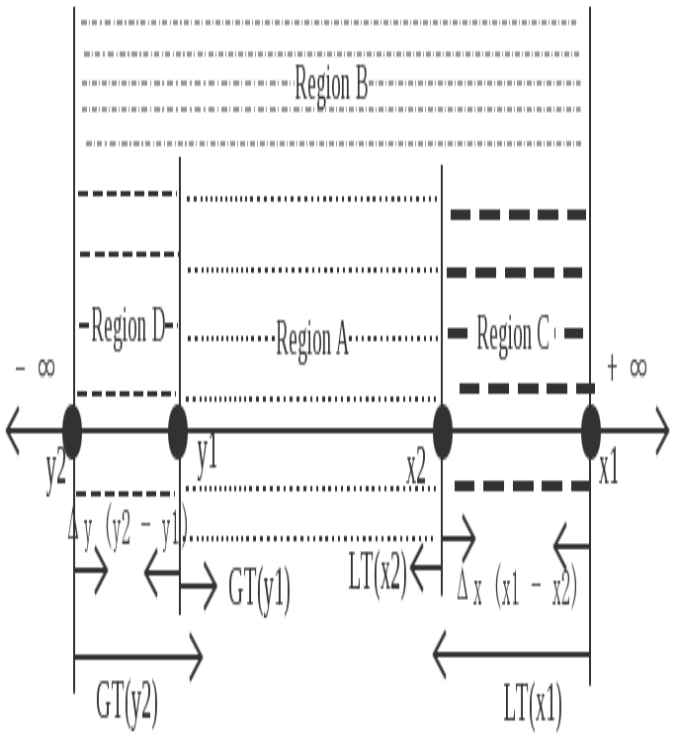


Figure 8: Range of values impacted by two pairs of LT-GT constraints in 2 ↔ 2 combination

3 Conclusion and Future Work

Some application domains such as a university environment use multiple databases which contain the same or similar data. For example, a Registrar's office in a university uses students information system using its own database, while a Financial Aid office in the same university may use another database on its own. This second database may have a lot of data elements that are the same or similar to the ones used by the Registrar's office. When multiple databases are used that contain the same or similar data, there is a potential problem due to inconsistency when updates to these databases are not properly synchronized.

In this paper, we proposed a method to consolidate two databases into one that can be used by both applications which previously used the two different databases. Our focus is on finding the commonalities and differences between the two databases and to let the user interactively select the data elements from both databases. In doing so, we needed to match data elements for their names, data types and associated constraints. Our process consists of three phases. Phase one is linguistic matching to match the names of two entities. This phase uses a knowledge base for keeping previously matched entries and some application domain information. The second phase uses graph theoretic approach to match structural elements. This phase is used to match schemas and subschemas. The third phase focuses on constraints imposed on the attributes and lets the user interactively select data elements from the two

databases. The final result is a single consolidated database that satisfies the constraints on data selected from both databases.

The work on linguistic matching and structural matching is fairly straight-forward. Our implementation on the third phase on constraint matching currently uses six logical operators that are commonly used on numeric data types (e.g., Less Than and Greater Than operators.) Accordingly, each attribute may have one to three constraints using these logical operators. This is quite complex because of the various combinations of these logical operators. The complexity proportionately increases with the number of constraints imposed on each attribute.

Though the paper describes examples using numeric data types only, the method is equally applicable to other data types such as strings, dates and so on. Our immediate future work is to focus on including other data types that are quite common in many databases. Current implementation used two different databases in a university setting. We would like to apply this method to other application domains as well. Another possible extension of this work is to analyze dependencies of constraints on data elements. For example, in a university setting, the financial aid allocated to a student will depend on the student's GPA. The current approach deals with only constraints applied to individual attributes, for example, GPA $\geq$ 3.5 and financial aid $\leq$ \$12,000. A dependency constraint in this case may require that financial aid for \$5000 and above can be awarded only if GPA $\geq$ 3.5. At the same time, a good academic standing status used by the Students Information system may require GPA $\geq$ 3.0 only. While matching the constraints on both databases, if the constraint GPA $\geq$ 3.5 is retained, a student who requires a financial assistance of \$5000 or more will be forced to maintain a 3.5 GPA for good academic standing as well. The matching process and interactive selection of attributes and their constraints then becomes more complex.

References

[1] Z. Bellahsene, A. Bonifati and E. Rahm (Eds.), *Schema Matching and Mapping*, Springer-Verlag, 2011.

[2] P. A. Bernstein, J. Madhavan and E. Rahm, "Generic Schema Matching, Ten Years Later", *Proceedings of the VLDB Endowment*, 4(11):695-701, 2001.

[3] B. Cate, V. Dalmau and P. G. Kolaitis, "Learning Schema Mappings", *Proceedings of the International Conference on Database Theory*, Berlin, Germany, pp. 182-195, March 26-28, 2012.

[4] N. Choi, I. Song and H. Han, "A Survey of Ontology Mapping", *SIGMOD Rec.*, 35(3):34-41, 2006.

[5] J. Euzenat and P. Valchey, "Similarity-based Ontology Alignment in OWL-Lite", *Proceedings of the European Conference on Artificial Intelligence (ECAI 2004)*, Valencia, Spain, pp. 333-337, August 22-27, 2004.

[6] K. Yadav and K. Periyasamy, "Constraint Matching in Multiple Databases", *Proceedings of the International Conference on Computers and Their Applications (CATA 2016)*, Las Vegas, NV, pp. 3-10, April 4-6, 2016.

[7] J. Madhavan, P. A. Bernstein and E. Rahm, "Generic Schema Matching With Cupid", Technical Report MSR-TR-2001-58, Microsoft Research, Microsoft Corporation, Redmond, WA, 2001.

[8] S. Melnik, H. Garcia-Molina and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching", *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, San Jose, CA, pp. 117-128, Feb 26-Mar 1, 2002.

[9] K. Periyasamy and P. Papanna, "A Tool for Schema Matching with a Knowledge Base", *Proceedings of the International Conference on Software Engineering and Data Engineering (SEDE 2014)*, New Orleans, LO, pp. 37-42, October 13-15, 2014.

[10] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching", *The VLDB Journal*, Springer-Verlag, 10:334-350, 2001.

[11] N. Rizopoulos, *Schema Matching and Schema Merging based on Uncertain Semantic Mappings*, Ph.D. Thesis, Department of Philosophy in Computing, University of London, Imperial College of Science, Technology and Medicine, November 2009.

[12] X. L. Sun and E. Rose, "Automated Schema Matching Techniques: An Exploratory Study", *Research Letters in Information and Mathematical Sciences*, 4:113-136, 2003.

[13] H. Q. Thang and V. S. Nam, "XML Schema Automatic Matching Solution", *International Journal of Electrical, Computer and Systems Engineering*, 4(1):68-74, 2010.

[14] P. Valchey and J. Euzenat, "Dissimilarity Measures for Collection of Objects and Values", *Lectures Notes in Computer Science*, 1280:259-272, 1997.

Kasi Periyasamy is a Professor in the Department of Computer Science at the University of Wisconsin-La Crosse, La Crosse, Wisconsin. He is also the Program Director for the Master of Software Engineering in the same department. His specialization is in Software Engineering, particularly on formal specifications, software reuse, software testing, cost estimation and software tool development for various applications. Recently, he has also been working on data science projects related to medical applications.

Karamveer Yadav is a graduate student in the Department of Computer Science at the University of Wisconsin-La Crosse, La Crosse, Wisconsin. He is working towards his Master of Software Engineering degree and is expected to graduate in May 2016. His research interests include Software Engineering and Data Science.

Designing Secure Computer Systems as Purposeful Systems

Maximilian M. Etschmaier* and Gordon Lee†
San Diego State University, San Diego, CA 92182 USA

Abstract

There exist many applications in which a system must be made secure, that is, one in which only authorized users may enter the system to submit inputs or take information out of the system. In this paper, we examine the requirements and performance that may be desired of secure systems in their design, analysis and operations. A single purpose computer is investigated as a case study and issues such as quality and configuration management are reviewed in the context of developing system requirements. A framework is suggested through which a group of secure single purpose computers can be integrated as nodes into a networked computer system and where the security properties of the nodes transfer directly to the network as a whole. A process is presented through which the detailed design and optimization of a system can be carried out. The principles of designing a secure computer system are based upon the case study of stand-alone and networked voting machines which is also presented to illustrate the process using purposeful systems.

Key Words: Purposeful systems, computer security.

1 Introduction

System security continues to be an important issue in many applications. Breaches in security include sabotage of an adversary's assets to gain a strategic advantage in war or economic competition or infiltration of settlements to create confusion and fear (the Trojan horse). The importance of secure systems dramatically increases due to ubiquity of information technology elements included in systems and the size of industry selling security solutions.

Unauthorized manipulation of the system may come in such forms as presenting to the system a false picture of the state of the environment, or in obtaining unauthorized information about the state of the system. There have been many approaches to protecting a system, and in particular, security of software systems (cyber-security). In [2, 12], for example, researchers have focused on the Rational Unified Process (RUP) which is a well-known software engineering process for assigning tasks and responsibilities within an organization. These approaches suggest the addition of roles and artifacts to reduce threats and improve security requirements. In [14], the author models activities based upon the Tropos methodology and adds a low level security-engineering ontology, derived from an UML approach. Using an open system analysis for system specifications, verification and synthesis, the authors in [13] model a system with a possible intruder and verify whether the whole system is secure, based upon a temporal logic formula that describes a secure behavior. Issues associated with firewall protection are presented in [11], while a survey of spam zombie detection methods is given in [3]. Security at the chip level is discussed in [9] using fault-tolerant network interfaces.

These and other current approaches generally focus on information technology systems and cyber-security. Many methods focus on the detection of unauthorized entry into the system, based upon user authentication (user name, password, security questions, biometric information, and/or user profiles and user statistics). Further, these methods detect attempts at unauthorized manipulation before it can be carried out (catch operational instructions as they are entered into the system - malware detection or detect unauthorized manipulation being executed within the system - system monitoring or plausibility checks). They may fortify information transmission (instructions and data) to and from the system to prevent alteration (encryption).

Yet, the issue of security continues to be an on-going issue and in [16], Schneier acknowledges that critical methods for securing systems still remain to be developed. Further, in [1], Alston and Campbell address secure systems from a systems engineering approach. A secure system is described as a set of functions and processes that protect a resource from threats, which is also the approach considered here.

A particularly difficult problem of assuring system security is posed by electronic voting machines. Because the ballot cast by any individual voter is supposed to be kept secret, there is no way to verify that during the actual voting process any ballot is correctly recorded. Etschmaier [4, 8] showed that security of a voting machine can be assured by properly structuring the internal architecture and operation of the voting machine as well as the interaction of the voting machine with the input-output devices and by embedding the voting machine in a precisely defined process of quality and configuration management throughout its entire life-cycle.

* College of Sciences. Email: metschmaier@mail.sdsu.edu.
† Dept. of Elec & Comp Engr. Email: glee@mail.sdsu.edu.

In this paper we present a generalization of the work on voting machines to a general purpose computer, following more explicitly the process of a purposeful system, which is defined by Etschmaier [5] in such a way that all aspects of the life-cycle can be considered holistically within the design process. System security is implicit throughout this process. We will return to the subject of voting machines in the form of a case study in Section 6.

2 A Purposeful System

A purposeful system was defined by Etschmaier [5] as a collection of functions and objects the boundary of which is chosen is such a way that the purpose of the system, to the extent possible, can be enclosed within it. The ability to enclose the purpose of any system within the system boundaries is limited because, except for the universe, any system is in some form of interdependence relationship with the environment. It is defined by the identification of objects that are part of it, and the characterization of the relationships between the objects and with the environment. Any human-designed system can be viewed as a purposeful system and the success of a system is the degree to which the purpose of the system is being fulfilled. This depends on its adaptability, agility, and nimbleness, as well as on the ability to effectively process intelligence.

The design of a purposeful system requires that the designer observe the interconnection of the system components and their functionalities, and assure that the functionalities, to the extent required by the system purpose, remain available throughout the life-cycle of the system. Thus a system design will search for system boundaries that can ensure that the system purpose can be met. As in the approach defined in [1], in a purposeful system, security, like operational availability and sustainment, are essential parts of the system purpose, not additions or enhancements. This requires that all aspects of system operations and sustainment, including maintenance and repair, as well as security are considered as integral parts of the system design process. Similarly, the definitions of the processes of quality and configuration management will evolve in the design process and extend over the entire life-cycle of the system.

Integration of the processes of quality and configuration management in the design process is a direct consequence of the perspective on time and evolution articulated in [5]. As shown in Figure 1, time is a continuous movement from the past to the future, with the only real point being the instant that is the present and represents the boundary between a past that is irretrievable lost and a future that is not yet and essentially unknowable. Knowledge about the past can only be obtained through contemporary records that were created and are analyzed by a "historian."

The analysis of the historian serves as the basis for control of the processes within the system and between the system and the environment. On the basis of the historian's analysis, the system creates possibilities for the future and the "system designer" adapts the design of the system to the changing

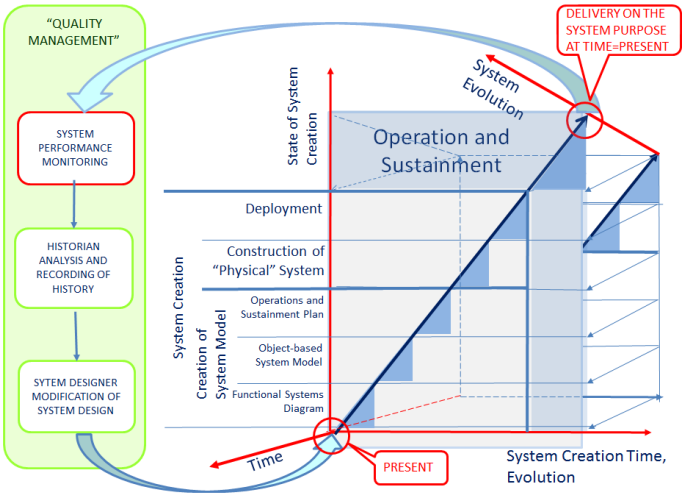


Figure 1: Life cycle of a purposeful system

environment ("evolution"). The historian and the designer are integral elements of the system and remain part of the system throughout its life cycle. If they are missing, the system loses its relevance and can no longer participate in the process of evolution. To a limited extent, the functions of the historian and the designer are often implemented through feedback and learning functions, although the importance of their presence throughout the life-cycle is often not appreciated.

The definition of system purpose and boundaries is not unique and the search for the best definition may require a circular and iterative search as security requirements are dependent on the definition of system boundaries, and system boundaries provide possibilities for security. The result of this process is a functional system diagram which may be employed in the design [14] as well to define system control for many operations that are based on the model of the system. The functional system diagram is used to visualize, in the most accessible manner, the essence of the system. It may be expressed in many different formats. Figure 2 shows the format we have found most useful to depict the structure of a voting machine.

As shown in Figure 1, development of the functional system diagram is followed by the development of a detailed, object-based model as the second phase of the design process. The object-based model serves as the basis of all remaining phases of the system life-cycle.

In this paper, a computer system is used to illustrate the processes of design, analysis and operations for secure systems. To do this, the functions of a computer system will be presented and the requirements on their availability will be discussed. Then threats such as design deficiency, malfunctions, violation of security (e.g., sabotage) which may impede the secure operation of the computer system will be presented. Based upon these threats and functions, key operations that assure that the design meets the requirements, manages design quality and quality of operation, as well as feedback and learning loops will be developed. The result of these processes provides a secure system which, in this

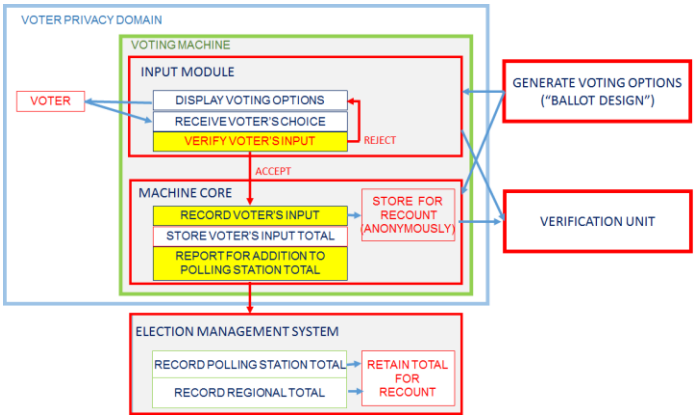


Figure 2: Definition of the structure of a secure voting machine

example study, is a computer system. An outline of the steps through which the detailed system design can be developed from the functional system diagram is provided in Section 5.

3 A Single Purpose Computer as a Purposeful System

A single purpose computer can be considered as a transfer machine. Its origins may be disputed but many agree that the architecture of a single purpose computer has its origins in the Turing machine. This machine can be modeled in many ways and is based upon transition states, inputs and outputs resulting in computational sequences. Ignoring the details of the execution of computational instructions, a highest-level representation of the functions and components of a simple computer is shown in Figure 3. The computer is divided into a processing unit which contains facilities for processing new input data and for preparing required output, and a storage facility which integrates and holds data for use by the output processing unit. It receives input in the form of data and processes them in a sequence of computations through various transition states into some form of output.

In such a configuration, there may be security risks that compromise the operations of the machine. Such risks arise from interaction of the computer with the environment.

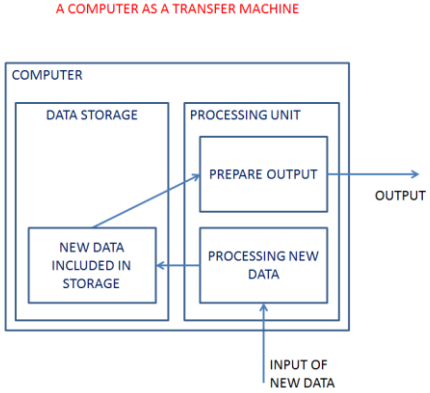


Figure 3: A computer as a transfer machine

Possibilities for such risks are readily identified and shown in Figure 4.

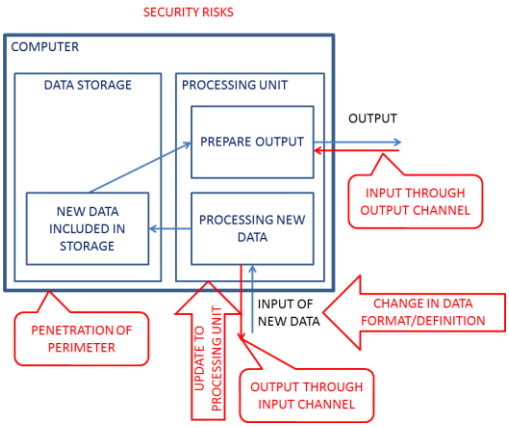


Figure 4: Examples of security risk points

Penetration of the computer perimeter may lead to a destruction or manipulation of stored data or of the storage unit. It may also lead to destruction or manipulation of the processing units as well as of programs stored in them. Damage to the processing units can also be inflicted through faulty updating of the programs that control them. The stored data can be rendered meaningless if, over time, the definition of the data or the format in which they are entered into the computer are inconsistent. Bad data may enter the computer through improper means, e.g., via the output channel. Finally, the security of the data may be compromised if through improper access, such as obtaining information from inside the computer through input channels, confidential or secret information is retrieved.

In order to investigate potential security risks, one may start with the internal organization of current computer systems; a simplified block diagram is shown in Figure 5.

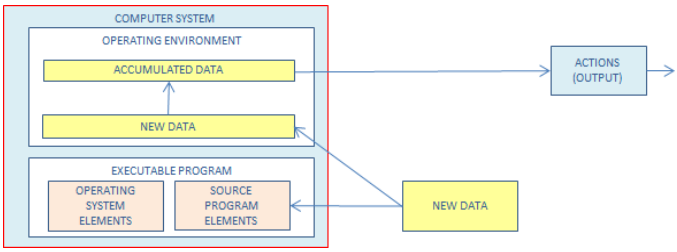


Figure 5: Internal organization in a functional system diagram

The computer is divided into a store of excecutable programs which holds the executable elements of the operating system as well as the executable source program elements; and the operating environment which holds the data in the various stages of processing. New data enter the computer in the form of additions to the store of new data as well as of control instructions to the source program elements. The output can take on several forms: it may simply present information, provide answers to queries, or control some form

of process.

With this functional system model, it is possible to address the two essential design questions: how to effectively and sytematically design a security process that can block incoming risks, and where to draw the system boundary.

4 A Framework for Security Assessment and Design

In this section we will translate the general concept of purposeful systems and functional system analysis to the design and operation for a secure machine; a single purpose computer will be used as an example. We will briefly describe the definition and key features of a system and then define a framework within which designer requirements must be satisfied.

Figure 6 shows possible points where external threats in the form of security leaks might enter the single purpose computer (system) in the form of malware and bad data, as well as a possible set of processes that one may define for protection against these threats. Malware is a collection of code elements that, alone or in combination with each other, can attach themselves to the store of code in the computer and activate processes that are outside the scope intended by the designer. Bad data are inert pieces of information that in format or definition deviate from what is expected by the designer. When added to the store of data inside the computer they will distort, possibly irreversibly, the information contained in the store of data, and thus lead to false output being delivered. Also, if bad data are allowed to enter the computer they may, in conjunction with faulty code, produce the same effect as malware.

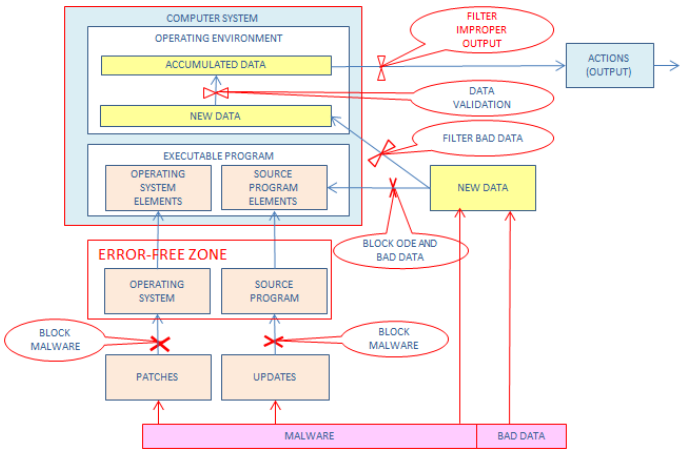


Figure 6: Security processes for a single purpose computer

In the example of a single purpose computer, analysis of reliability requirements [4, 8] leads to the definition of a system as a secure repository. This single purpose computer can hold the data entered into it. It feeds into the higher layers of the system but does not interact with them. Separately, there may be processes through which, from time to time, the operating system and the source program may be modified. These processes, carried out by the system designer, essentially amount to redesign of the system and require revisiting all aspects of the design and operational processes as recorded and documented (analyzed) by the historian. They are separate from each other and independent of the flow of data. Protection of the integrity of the system requires that authentic copies of the latest version of the operating system and the source program are available and that modifications are protected against improper manipulation. While the operating system and the source code do not need to be part of the computer system, they need to be stored in an error-free zone and, when needed, securely connected to the computer. Any input of patches or updates needs to be screened for malware and errors.

Examination of the diagram permits immediate identification of where what security functions need to be included in the system to assure that these processes do not interact or otherwise cross their boundaries.

The important aspects of the single purpose computer are that: (i) it is a clearly defined product with clear boundaries; (ii) it consists of hardware, software, and data storage elements; (iii) the hardware and software are designed not to change over the life of the machine (any modification will create what may be considered an essentially new system); (iv) data held in the storage devices only change in narrowly defined ways and without penetration of the perimeter of the machine; (v) it requires no routine maintenance, and its functions can be diagnosed from the outside; and (vi) data can be entered and accessed without disturbing the perimeter of the machine.

Figure 7 shows a generic model of a single purpose computer where one may focus on the user interface to create an outer layer of security. The model is neutral as to the technology used to build an actual machine. This single purpose computer can be developed and produced in an environment where there are precisely defined rules and requirements, where the required functions and their criticality are precisely defined, and the rules are enforced unambiguously. The figure also shows how the user and the functions of the history file, the historian, and the designer (defined in Figure 1) are included in the system. The history file continuously receives information about the output from, or actions commanded by the computer system as well as from the actions of the user or the performance of the controlled process. The content of the history file is analyzed by the historian and integrated into the history of the system. Based on the analysis of the historian, the designer identifies, develops and implements modifications of the system that appear desirable or necessary. The modifications are implemented through updates to the source program. Patches to the operating system are provided by the supplier, preferably screened by the designer. Access to the history file may also be provided to the user or the controlled process.

As the historian and the designer have critical roles in the maintenance of the integrity of the computer system, they are included within an outer security perimeter which also includes all elements modifying the operating system and the source code. Separate from the outer security perimeter is the user domain which includes all elements of the user interface

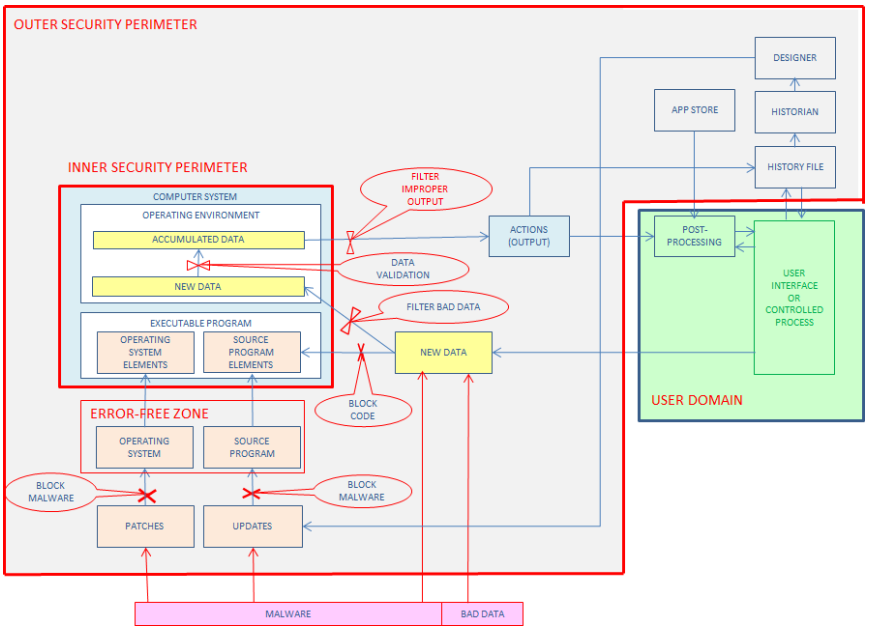


Figure 7: Designing a secure system, creating an outer layer of security

or of the controlled process as well as the post-processing activities required to connect to them. The user domain may have its own security requirements. But those are independent of the security requirements for the computer system. The post-processing activities may, however, (when necessary) fetch program elements from an app store that is included in the outer security parameter of the computer system.

The design of a secure system may be modeled as a string of nodes within a design process. In Figure 8, the functional system diagram of the single purpose computer is arranged in such a way that it can be incorporated as a node in such a network. There are two ways to construct a network: in a network with decentralized (parallel) processing, each node may replicate all system functions with only the store of accumulated data being assigned to a central function; alternatively, in a network with central processing, the nodes may only incorporate the user domain with one computer system serving all nodes in addition to providing the central storage of accumulated data. For both alternatives, essential security functions transfer directly from the nodes to the networked system.

Figure 9 shows the first and last nodes of the first alternative of a networked computer system. Sharing of functions within a fully functional secure network may be quite limited. For there to be a meaningful network, clearly the function of the designer and at least some functions of the historian need to cover the entire network. Beyond that, sharing may be limited to some aspects of the user domain and the store of accumulated data. The store of accumulated data represents the element that actually integrates the network. It requires the highest level of protection in the network because bad data entered into it will corrupt the accumulated data and may cause irreversible damage; and that damage might go unnoticed for a long time, if not forever. For this reason, this store is enclosed by an inner security perimeter that is shared by all nodes. Additionally, these data may be organized in a structured way

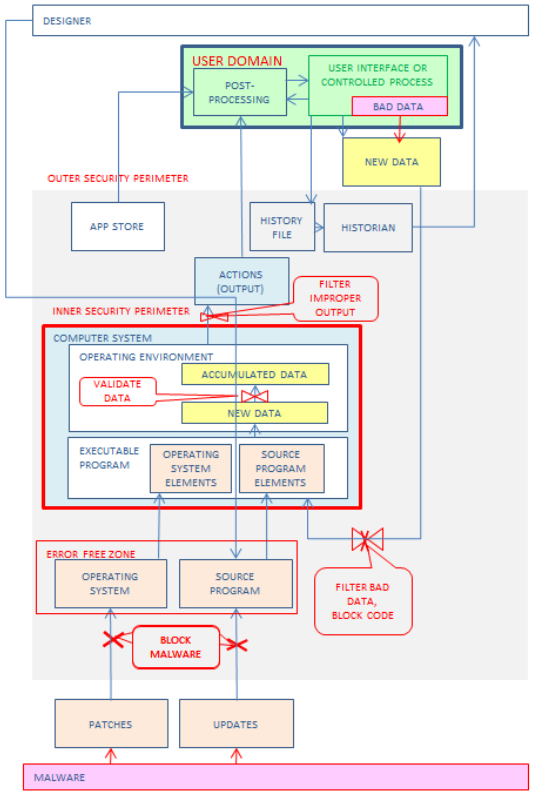


Figure 8: An example of a node within the network of nodes representing security points

and the access of each node limited to select strata of data.

Figure 10 shows the arrangement of a networked system with central processing. The nodes are formed by the relatively simple user domains. However, the possible cost saving from this is balanced by the need for increased data exchange between each node and the central computer.

Vulnerability for both alternatives arises from the transmission of data between the nodes and the central element of the system. For decentralized processing, it is the transmission between each node and the central data storage unit. For central processing, it is the transmission of information between each user domain and the central

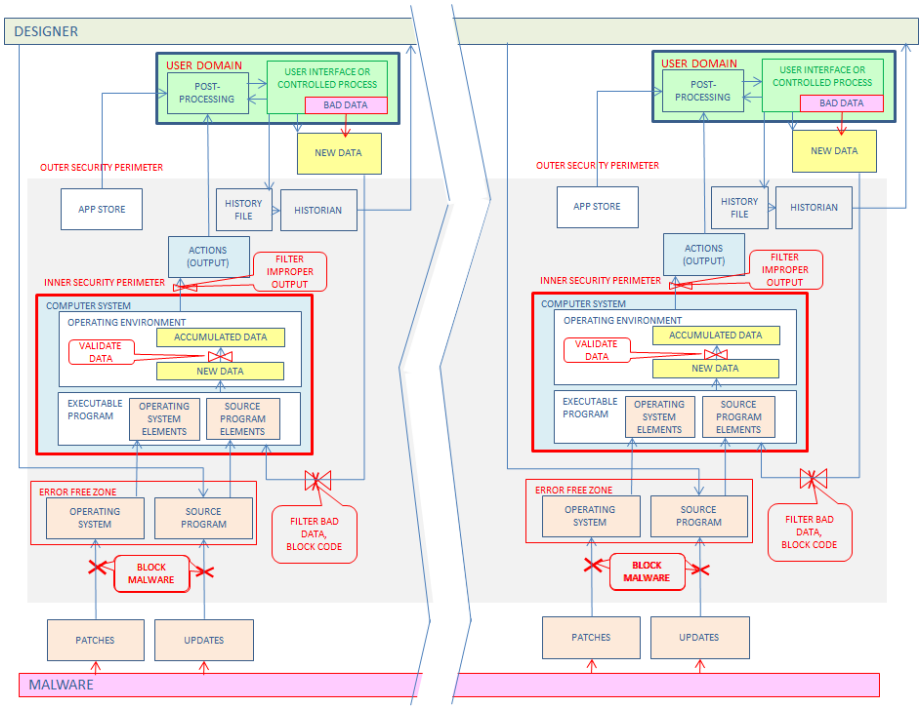


Figure 9: System security division in a networked system with decentralized processing

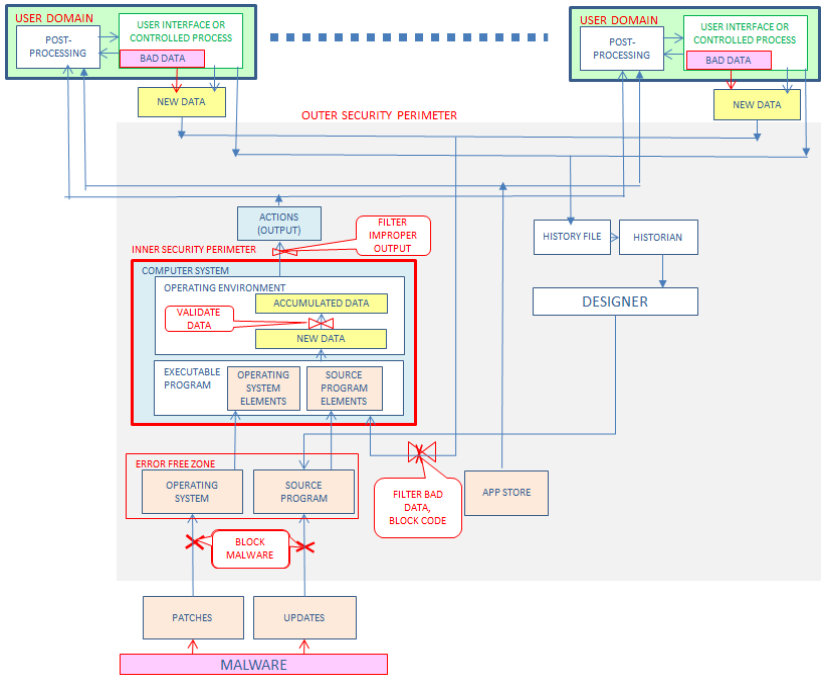


Figure 10: System security division in a networked system with central processing

computer system. In order to assure security, both cases require secure transmission of data.

5 Realization: Design, Operation, and Sustainment

A functional system diagram articulates the essence of the computer system. It defines the functionality, identifies the importance of each function, and assigns the functions to a system framework. It represents a top-level system specification. Following [6], it can be used as the basis for developing the detailed design, and the plan for construction, as well as for operation and sustainment of the system. With the SOMPA model one can turn the principles of design spelled out in the functional system diagram into physical and logical elements, study the full complexity of their behavior under condition of the real world and develop in detail the mechanisms that will provide the defense against threats to security (and failures, breakdowns). A top level view of SOMPA is shown in Figure 11 [6]. As the principal tool of the process of physical design, SOMPA also permits explicit optimization over design alternatives. SOMPA thus provides an alternative over the established processes of designing computers and in particular software. It is more flexible and intuitive.

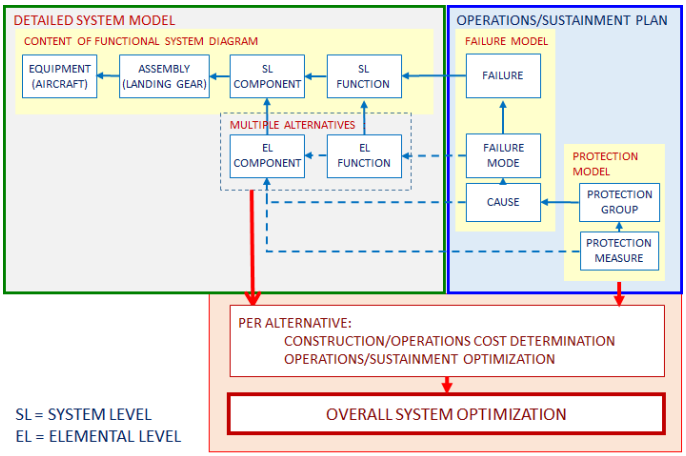


Figure 11: Overview of the SOMPA model

SOMPA is a model that represents a system as a complex network of objects. It uses objects to represent all aspects of the system, system components as well as functions, failures, operational and sustainment activities, and selected system properties (like system states). Every object is an independent entity that is linked to other objects through changing relationships. SOMPA permits the use of any object in multiple places of the network at the same time as well as the creation of multiple objects from which to choose for any position in the network. This makes it possible to create a variety over which the system can be optimized. The possibilities for optimization concern design choices as well as strategies for system operation and sustainment.

Any object used in the SOMPA model can incorporate any information that might pertain to it over all or parts of its lifetime. Consequently, the model can be configured to be a simile of the physical system it represents. As shown in Figure 12, it can be linked to the physical system to provide a real-time simulation of the processes occurring in the physical system and the transformations that system may be undergoing. Control measures can be developed in the model and applied to the physical system. Comparison of the model states with the states of the physical system can identify emerging and progressing system failures. Such failures may be caused by malfunction of system components, by errors in the design of the system, or of its components, including the human element, or by errors in the model formulation and of the control commands that emanate from it. Even as the physical system is built, operated and sustained following the SOMPA model, it may differ from it in many respects. Those differences may be due to human errors in the construction of the physical system, or they may be due to the fact that certain aspects of the behavior of the physical system might not be understood by the creators of the physical system, or simply be beyond the state of the art, i.e., could not be known at the time of creation of the physical system. Using the SOMPA system to control the physical system in real time will thus provide the critical capability to sustain the system into an indefinite future. In terms of Figure 1, it provides the information needed for the analysis function of the historian and the continuous redesign of the system by the designer.

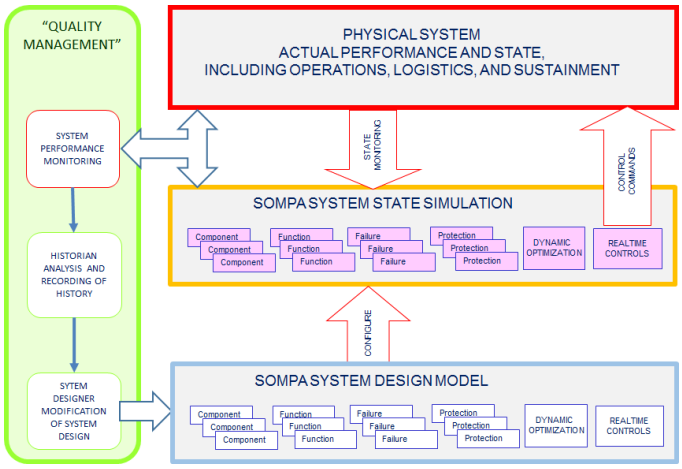


Figure 12: Using the SOMPA model to control the physical system

While SOMPA was originally developed to control manufacturing machines in the context of the NIST Smart Machining Program [7], the object-based structure makes it applicable to any type of system, including computer systems that combine hardware and software elements. The functional system diagram of Figure 8 articulates the vulnerability of the basic architecture of a simple computer. It shows a hierarchy of security perimeters, an isolated user interface module, and censor-type functions that can protect the integrity of the data and code elements. The simple computer presents as a

mechanistic device that executes logic operations through a collection of algorithms. Figure 8 does not distinguish between hardware and software elements. In fact, many of the elements can be realized either as hardware or as software elements. The optimization processes of SOMPA can readily identify the best realization just as they can choose between different components or different operational or sustainment strategies.

SOMPA thus provides a tool through which the hardware as well as the software element of a computer system can be fully modularized. In the design process each module can be designed separately (or selected from a set of readily available products). The functionality and performance of each module can be verified separately. SOMPA can optimally select between alternative available modules, and define the necessary interaction capability between them. SOMPA can also function as a tool to control the operation and sustain the computer throughout its lifetime. Thus, SOMPA can assure that the entire computer system is designed to meet all requirements and errors in the design are eliminated to the maximum extent possible. In particular, the computer system will be protected against "critical" failures. Using SOMPA to control the operation and sustainment of the computer will assure that any errors that might remain will be recognized and kept from causing "critical" failures.

6 Case Studies

The design of a secure computer system is based upon previous work on the design of a secure voting machine. In this section, a brief discussion of the case study of a voting machine is presented for completeness and as another case study, to illustrate the process of secure system design using purposeful systems. It is this framework that can then be applied to the design an analysis of secure computer systems.

6.1 A Stand-Alone Voting Machine

Voting machines are arguably the most important physical tools of a democracy. They provide indiscriminate access to the election process, tabulate unambiguously the outcome of an election, and guard the confidentiality of the voting process. Malfunctioning voting machines may change the outcome of an election. As elections may decide between peace and harmony, or war and turmoil, improperly functioning voting machines may literally put many lives at risk.

Voting in an election is a relatively straight-forward matter: a voter selects from slates of candidates or issues and makes the choice known to some election manager who tallies the votes and determines the winning candidates or issues. This process can easily be carried out within an assembly of voters by raising hands as the various issues are called. If not all voters can be physically present in a polling place, they may record their choices by marking a ballot. If a person's choices are to be confidential (or secret), the ballots may be placed in a ballot box together with ballots of other voters and removed randomly for counting. A desire to mechanize this process appears to be motivated by several expectations: mechanization would make it easier to tally the votes; it would provide a vehicle for the voters to unambiguously express and record their choices; it would make it impossible to alter the collection of ballots cast by adding or removing ballots.

The earliest efforts at mechanizing the voting process appear to be dating to the mid-19th Century [18]. Over time, various technologies for voting evolved and voting machines became widely used throughout the US. However, not only did none of the designs ever fully meet expectations, but their limited reliability added new uncertainty to the election process. Also, lack of funds in many jurisdictions limited the ability to procure adequate numbers of voting machines, leading to long lines at some polling places, and ultimately denial of access to the voting process. The problems of voting machines came to a head in the 2000 US presidential election when disputes over the validity of punch-card ballots in one Florida precinct led to weeks of legal wrangling, and the outcome of the election ultimately was decreed by an act of the US Supreme Court.

To avoid a repeat of the 2000 election, the US Congress passed the Help America Vote Act (HAVA), mandated the National Institute of Standards and Technology (NIST) to undertake a program to define standards for voting machines, and appropriated more than $2B to fund the acquisition of new generations of voting machines throughout the country and established the Election Assistance Commission (EAC) to oversee these processes [17]. However, a recent study by the prestigious Brennan Center for Justice [15], fifteen years after HAVA, is quoting an EAC official that "Our voting equipment is old and past its usefulness. We're getting by with Band-Aids, but I worry about a crisis with some of the older machines." And it observes that "flipped votes, freezes, shut downs, long lines, and, in the worst case scenarios, lost votes and erroneous tallies … have … been occurring more frequently than they should." It also points out that "It is worth remembering that the machine problems in Florida in 2000 could have gone unnoticed … had they not happened in an exceptionally close election."

Rather than by embarking on yet another round of investing in new generations of equipment that merely extends existing design principles, work that started as part of the NIST Voting Program showed that the problems may be solved once and for all by following the concept of purposeful systems and applying the holistic design principles in the design, manufacture and operation and sustainment to design an entirely new generation of voting machines [4, 8]. Such voting machines could meet all expectations of reliability and security at a cost below that for current generations of voting machines. The case study on voting machines outlines how a standard can be formulated that will assure that systems such as voting machines can meet expectations. We start the discussion with a stand-alone machine and extend it to a system for an entire precinct. Finally, we show how this standard can be extended to cover internet-based voting.

The purpose of a stand-alone voting machine is to serve as a secure receptacle within which a voter can deposit his/her vote without being observed, to tally the votes cast, and to provide a

record for verifying the tally in a recount. The top-level functions required of a voting machine can be presented as in the functional system diagram of Figure 2. The part that corresponds to a stand-alone voting machine is contained in the box labeled "voting machine." Examination of the functional system diagram shows that operation of the voting machine consists of a sequence of mechanistic processes that may be implemented through a collection of purely mechanical devices. In this discussion it is assumed that many of these processes are realized in terms of algorithms that are implemented in computer technology. Through those processes security and reliability may be compromised through faulty design of the algorithms and their interaction with each other, as well as through failure of the physical elements through which the algorithms are implemented. Clearly, there is no essential difference between these failures and the failures of any other component of the voting machine. Thus the process outlined in Section 5 is fully applicable to the design of a voting machine. Any standard for voting machines can be based on it.

Figure 13 shows the list of potential failures, the "critical failures," that must not be permitted to occur if the voting machine is to meet its purpose. As a first step to satisfying this security requirement, the functions are divided between an input/output module and a machine core, the principal components of the voting machine. This arrangement maps directly into the components identified in Figure 3 for the simple computer, with the machine core accommodating the processing of output in addition to the inclusion of new data in storage. The input of the ballot is handled by the input module and occurs in two directions. In order to avoid the risk of bidirectional channels, this requires two separate channels, one through which the computer presents the voting options, and the other through which the completed ballot is received.

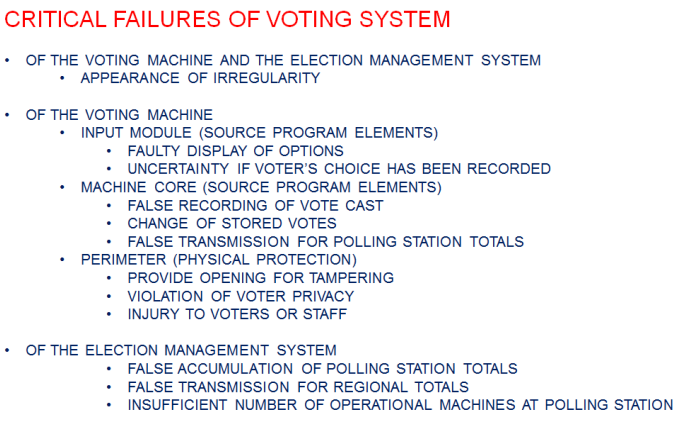


Figure 13: Potential critical failures of a voting machine

The input module does not retain any ballot information, but passes every ballot, after it has been received, on to the machine core for inclusion in the core memory. In order to keep this memory from being contaminated, faulty ballots have to be identified so that they can be recognized in the memory as that and entered to the collection of invalid ballots. The verification of the ballot is done by the input module which may or may not provide the voter with an identification of the defect through the same channel the ballot options were displayed, and provide him/her with an option to submit a corrected ballot.

The voting options available to the voter vary from election to election as well as, possibly, from precinct to precinct. To make a voting machine model usable in multiple elections and precincts, information for creating the options has to be entered into the machine from the outside. This requires that the machine is connected to an outside entity that supplies the necessary information. In order to protect the integrity of the programs stored in the voting machine, the corresponding input channel must not be permitted to carry any code. This means the generation of voting options needs to be divided into code that is permanently stored inside the voting machine, and inert data that will drive the code to generate the ballot information in a way that it can be transmitted to the voter. For this purpose, code elements will be included in the input module where they set up the ballot generation as well as the voter's input verification processes; and the machine core where they will set up the storage facility for the input total as well as for the storage of the anonymous ballots that might be required for an orderly recount. Generation of voting options is located in a separate unit which provides data to many voting machines, and is attached to a voting machine through a secure, severable link.

Corresponding to the quality management system shown in Figure 1, the voting machine needs to be equipped with some form of capability to monitor its performance. The verification unit which is shown in the functional system diagram in Figure 2 is configured as an external device that can be attached to many voting machines. The device calls up information from inside the voting machine, including of the data stored for a possible recount, but is configured in such a way that communication is limited to the transmission of inert data, and there are separate channels for input and output of these data. Code elements to perform the verification and performance monitoring are divided between both modules in the inside of the voting machine and the verification unit.

Most risks of critical failures of functions associated with components inside the voting machine concern failures of the programs stored in them. To protect against such failures two conditions, have to be met: The programs have to be free of errors, and the possibility of manipulating them from the outside has to be eliminated. The second condition requires that the input module and the machine core each need to be isolated from each other logically and physically, and the voting machine as a whole needs to be enclosed by an impenetrable enclosure. This enclosure also needs to protect humans and election officials and other persons from injury.

To protect against violation of voter privacy, a separate perimeter needs to be established that encloses the voting machine together with the voter for the duration of the voting process. Output from the voting machine is limited to transmission of the vote totals at the end of the election.

The approach discussed in this case study differs significantly from the current approach that assume that the core of a voting machine would be an unmodified general purpose computer ("COTS") running a commercially available operating system, like Windows, to execute software that meets the standards for general commercial-grade programs. Such standards limit, but do not exclude the occurrence of programming errors. The current approach is to achieve security (and secrecy) through definition of standards for design and performance of hardware components and through obscuring information handled by the voting machine through encryption. It attempts to demonstrate security (and reliability) by extensive testing through simulated voting. However, error free performance of a voting machine during testing, when secrecy is not required, does not provide any assurance of error free performance during an election when security is required and ballots cast cannot be compared tabulated results. For example, testing could not detect code that would be executed during regular voting but not during the test, such as what has been found in vehicle emissions control systems [10].

6.2 A Precinct Network

We have shown that the design of a stand-alone voting machine for security and reliability follows closely what we have described for a simple computer. As shown in Figure 9, a number of stand-alone voting machines, such as the one located at the same precinct, can be connected by sharing the store of information that is common to all. The shared memory in this case is the record of the polling station total, the first function of the election management system. This leaves the security requirements for the individual voting machines unchanged, except that security for the output channel has to be provided during the entire duration of the election.

Alternatively, following Figure 10, a network can be configured with central processing. In this case the nodes would be formed only by the input module, while the machine core and the precinct-based part of the election management system, recording the polling station total, form the hub of the network and can be protected by the same enclosure. This arrangement breaks up the voting machine perimeter and requires increased levels of security for the transmission of data between the input module and the central hub. The distance for the transmission is increased significantly. Also, arrangements have to be made for the unit that generates voting options and the verification unit to access the input module separate from the central hub.

7 Conclusions

This paper has shown how the concept of a purposeful system together with the functional systems analysis process can be used to define and meet the essential security requirements for a single-purpose computer and how such a computer can be included as a node in a networked computer system. The essential security functions transfer directly from the nodes to the networked system.

The paper has also shown that the definition of functions of system operations, sustainment, and security need to be integral parts of the design process, and that the design process does not end with the start of the operational phase. To make this possible, a historian and a designer need to be and remain integral parts of the system throughout the entire system life-cycle. Processes of quality and configuration management, properly implemented, will be part of the functions of the historian and the designer.

The example of a voting machine system is used to show how the principles defined for the simple computer has been applied to a real system of considerable importance where security is of utmost concern. Development of a system from well-defined functions and components and assembly of a large system from independent nodes has been shown to improve transparency, simplify programming, and make it possible to integrate security measures directly into the design architecture.

References

[1] I. Alston and S. Campbell, "A Systems Engineering Approach for Security System Design," *Proc. of the International Conference on Emerging Security Technologies*, pp. 107-112, 2010.

[2] C. E. De Barros Paes and C. M. Hirata, "RUP Extension for the Development of Secure Systems," *Proc. of the International Conference on Information Technology-New Generations*, pp. 643-652, 2007.

[3] Z. Duan, P. Chen, F. Sanchez, Y. Dong, M. Stephenson, and J. M. Barker, "Detecting Spam Zombies by Monitoring Outgoing Messages," *IEEE Transactions on Dependable and Secure Computing*, 9(2):198-210, 2012.

[4] M. M. Etschmaier and Gordon K. Lee, "Regulating Product Security in the Public Domain: The Example of Voting Machines," Unpublished Manuscript, 2015

[5] M. M. Etschmaier, "Purposeful Systems: A Conceptual Framework for System Design, Analysis, and Operation," *International Journal for Computers and their Application*, 22(2):1-13, June 2015

[6] M. M. Etschmaier, S. H. Rubin and G. Lee, "On the Use of SOMPA Core Modeling for Systems Design," *Proc. of the World Automation Congress*, Kona, Hawaii, LNCS, pp. 1-5, August 2014.

[7] M. M. Etschmaier, *Developing Maintenance Programs for Smart Machining Systems*, Collection of Reports Produced to Support the NIST Smart Machines Program, GME International Corporation, June 2007

[8] M. M. Etschmaier, *Voting Machines Can Be Reliable, Trustworthy, and Affordable*, Collection of Reports, Produced to Support the 2007 Revision of the Voluntary Voting System Guidelines (VVSG2007), GME International Corporation, June 2007

[9] L. Fiorin and M. Sami, "Fault-Tolerant Network Interfaces for Networks-on-Chips," *IEEE Transactions on Dependable and Secure Computing*, 11(1):16-29,

2014.
[10] Danny Hakim and Keith Bradshersept, *After Volkswagen Revelation, Auto Emissions Tests Come Under Global Scrutiny*, The New York Times, September 24, 2015
[11] H. Hu, S. J. Ahn and K. Kulkarni, "Detecting and Resolving Firewall Policy Anomalies," *IEEE Transactions on Dependable and Secure Computing*, 9(3):318-331, 2012.
[12] P. Jaferian, G. S. Elahi, R. A. Mohammad and B. Sadeghian, "RUPSec: Extending Business Modeling and Requirements Disciplines of RUP for Developing Secure Systems," *Proc. of the 31st EUROMICRO Conference*, 2005:232-239, 2005.
[13] F. Martinelli and I. Matteucci, "An Approach for the Specification, Verification and Synthesis of Secure Systems," *Electronic Notes in Theoretical Computer Science*, 168(SPEC. ISS):29-43, 2007.
[14] H. Mouratidis, J. Jürjens, and J. Fox, "Towards a Comprehensive Framework for Secure Systems Development," *Lecture Notes in Computer Science*, LNCS, 4001:48-62, 2006.
[15] Lawrence Nordon and Christopher Famigetti, *America's Voting Machines at Risk*, Brennan Center for Justice, New York, NY, 2015)
[16] B. Schneier, "The Death of Security Industry," IEEE Security and Privacy, p. 78, 2007.
[17] United States Congress, *Help America Vote Act*, Public Law, 107th Congress, 107–252—October. 29, 2002
[18] Wikipedia, Voting Machine, https://en.wikipedia.org/wiki/Voting_machine, accessed 4-20-2016.

Maximilian M. Etschmaier is a native of Austria. He holds a Doctorate in Engineering from the Technical University in Graz, Austria, and an MS in Operations Research from Case Western Reserve University, where he was a Fulbright Scholar.

He is an Adjunct Professor at San Diego State University and also principal of GME International Corporation, where he has been advising clients on policy and strategy development, leading system development and process improvement ventures, and supporting international technology transfer. Previous positions include Chairman of the Management Board of Joanneum Research in Graz, Austria, Professor of Engineering at the Universities of Pittsburgh and Massachusetts, Senior Scientist at the United Technologies Research Center, Head of Operations Research of Deutsche Lufthansa AG, and Visiting Professor at the University of Graz and University of Innsbruck.

Dr. Etschmaier's professional work is focused on the design, analysis, and operation of complex systems in a wide variety of domains. The present paper reports on one of several case studies he has developed with Dr. Gordon Lee for this purpose.

Gordon K. Lee was born and raised in Hawaii. He received his B.S. degree in Electrical Engineering from the University of Hawaii in 1972, his M.S.E.E. degree from the University of Connecticut in 1974 and his Ph.D. degree from the University of Connecticut in 1978. From 1978 through 1989, Dr. Lee was at Colorado State University in the Department of Electrical Engineering where he rose to the level of Full Professor. He was also the Director of the Institute for Robotic Studies.

In 1989, Dr. Lee became a faculty member in the Department of Mechanical and Aerospace Engineering at North Carolina State University and also served as Director of Graduate Programs in the Department of Mechanical and Aerospace Engineering and later as Assistant Dean for Research Programs in the College of Engineering. Dr. Lee joined San Diego State University in December 2000 where he served as the Associate Dean and Director of the Joint Doctoral Program for the College of Engineering. He was also a full Professor in the Department of Electrical and Computer Engineering and is currently Professor Emeritus in that department.

His research interests are in the areas of robotics and intelligent control systems, particularly evolutionary control algorithms, fuzzy systems and neural networks, as well as in the applications of these methods to mobile robotic colonies. His research projects have been funded by government agencies as well as industry. He has published over 275 technical documents; Dr. Lee is a senior member of IEEE, a member of AIAA and a senior member of ISCA. He is also currently an Associate Editor for the International Journal on Intelligent Automation and Soft Computing.

Server Mechanisms for Guaranteeing Schedulability with RTOS Processing and Improving Application Responsiveness by Slack Reclaiming

Kazuki Hasegawa* and Kiyofumi Tanaka*
Japan Advanced Institute of Science and Technology (JAIST)
Asahidai, Nomi, Ishikawa, 923-1292 JAPAN

Abstract

With a great diversity of real-time embedded systems, developers often utilize a real-time operating system (RTOS) to efficiently build the systems. However, in exchange for the usefulness, execution of RTOS codes brings runtime overhead, which can influence the system schedulability. In this paper, we propose techniques for guaranteeing schedulability including RTOS runtime overhead. In the proposed techniques, we introduce two servers with planned capacity (or budget): periodic management server and aperiodic management server, which are dedicated to RTOS processing. The former takes charge of invoking periodic tasks at their period timing and the latter manages aperiodic RTOS events such as task completion. Then, we provide the schedulability test considering the servers' utilization. In addition, we show techniques for slack reclaiming to improve the aperiodic application tasks' responsiveness by utilizing capacity (or slack) of the servers which is left unused. The evaluation by simulations with synthetic task sets shows that average response times for aperiodic application tasks can be improved by up to 18.3% while guaranteeing schedulability of periodic tasks, and that the effectiveness of the slack reclaiming is more significant when a system tick is shorter for higher degree of precision.

Key Words: RTOS; real-time scheduling; polling server; deferrable server; slack.

1 Introduction

In these days, there is a growing demand for complicated real-time embedded systems such as mixed-criticality systems [1, 3] and systems where various types of real-time tasks coexist are increasing. It is often efficient to build such complicated systems with real-time operating systems (RTOS). Use of RTOS is helpful in that developers do not have to describe codes for task management including task scheduling and context switching, memory management, synchronization/communication, time management, etc. Instead, developers can be devoted to describing algorithms for their applications. However, in exchange for the usefulness, execution of RTOS codes brings runtime overhead for its sophisticated mechanisms.

Especially in hard real-time systems, it is important to guarantee the schedulability of all the hard tasks. As a representative method of assuring schedulability, one which takes processor utilization by tasks into account has been established and used, where the sum of the tasks' utilization is kept equal to or lower than the least upper bound of the task set (*Ulub*) [2]. However, it should not be sufficient to consider schedulability only with utilization by application tasks. This is because execution overhead of the RTOS codes including task/context switching, queue manipulation, or timer interrupt handling cannot be ignored. The existence of RTOS overhead can make the system unschedulable.

It is possible to estimate RTOS overhead and take them into account for schedulability analysis [2, 4]. However, the estimation about the frequency of preemption or context switching can be done only for periodic tasks which are invoked at regular intervals. When aperiodic (not sporadic) tasks exist, it is not possible to assure that actual RTOS overhead is within the estimated amount. In addition, the analysis model depends on an ideal assumption that every single event can occur only at a multiple of the system tick. It cannot be expected that aperiodic events and a system tick synchronize since the system tick is relatively long (from hundreds of microseconds to a millisecond).

In this paper, we propose techniques for guaranteeing schedulability, including RTOS overhead, for models where aperiodic RTOS events as well as periodic ones exit and aperiodic events happen asynchronously with the system tick. In the proposed techniques, we introduce two servers, periodic management server and aperiodic management server, dedicated to RTOS processing and guarantee the system schedulability involving RTOS overhead. In addition, we improve the aperiodic application tasks' responsiveness by utilizing unused capacity (or slack) of the servers.

This paper consists of six sections. Section 2 describes related works for scheduling algorithms, especially for task sets with hard, periodic tasks and soft (or non-real-time), aperiodic ones. Then Section 3 proposes two management servers for RTOS processing to guarantee the schedulability. Additional slack reclaiming technique for improving application tasks' responsiveness is proposed in Section 4. Evaluation of the proposed technique is shown in Section 5. Finally, Section 6

* School of Advanced Science and Technology. Email: {kazuki_h, kiyofumi}@jaist.ac.jp

concludes the paper with summary and future work.

2 Related Work

The proposed technique is based on Rate Monotonic (RM) scheduling [7], where every task has a fixed priority according to its period. RM has the merits of low implementation complexity and small jitters for high-priority tasks. There are RM-based server algorithms that provide aperiodic tasks, which do not have their periods, with priorities for the execution order. As representative examples, there are Polling Server [5], Deferrable Server [5], Priority Exchange [5], Sporadic Server [8], and Slack Stealing [6]. These servers aim to keep response times of aperiodic tasks as short as possible while maintaining schedulability of hard periodic tasks.

Polling Server has its period and capacity. Its priority is decided with the period in the RM context. In its turn, it serves aperiodic tasks as long as capacity is left. The capacity is decreased by the same amount as the time for which aperiodic tasks are executed. When the server's turn comes while there are no aperiodic tasks to be executed, the capacity is abandoned immediately.

Compared to Polling Server, Deferrable Server is an algorithm which improves aperiodic responsiveness by preserving its capacity even when there are not aperiodic requests pending. However, the better responsiveness is derived at the expense of lower least upper bound for the utilization, that is, maximum utilization by application tasks has to be lower than that in the case of Polling Server. The method in this paper achieves schedulability involving RTOS overhead by using Polling and Deferrable Servers.

Priority Exchange is an algorithm where, when there are no aperiodic requests pending, the server capacity is deposited in the lower-priority of other periodic tasks executed at the moment and can be subsequently consumed at the (lower) priority when aperiodic requests arrive. Due to this feature of lowered priority, Priority Exchange cannot be applied in the proposed method since RTOS processing including task activation, task termination, and context switching should be performed at the highest priority every time.

Sporadic Server and Slack Stealing are more complicated algorithms than those mentioned above while shortening response times of aperiodic tasks further. In Sporadic Server, the server capacity is replenished at different timing from system ticks, which makes it difficult to apply this server for periodic RTOS events such as invocations of periodic tasks. As for Slack Stealing, it imposes unrealistic complexity and memory requirement. The complexity requires large implementation or runtime overhead. The comparison between server algorithms in terms of computation/implementation complexity is found in [2]. Hence, these are not employed in our method.

3 Servers for RTOS Processing

In this section, a method of guaranteeing schedulability including RTOS processing is proposed.

In the system model, there are two tick units: system tick and processing tick. The former is units of time to express time at which periodic tasks can be requested to run and time duration for which tasks are supposed to run, that is, worst-case execution time (WCET). The latter is units of time to express the other time properties; tasks' actual execution time, actual finishing time, and RTOS's processing time.

Two servers for RTOS processing are deployed in the system. One is a periodic management server which takes charge of invocation processing of tasks and, if necessary, task switching. The other is an aperiodic management server which processes aperiodic events such as tasks' completion and task switching. These two servers are supposed to have the shortest period and the highest priority in the RM context since the RTOS processing should be performed in priority to other application tasks' processing.

3.1 Periodic Management Server

Periodic tasks are supposed to be requested in synchronization with system ticks. For example, a periodic task with period of four system ticks is requested every four system ticks. When this task is included in the system, the periodic management server (PMS) processes the invocation every four system ticks. On each system tick timing, there can be invocations of two or more tasks. PMS exploits the Polling Server algorithm and is given capacity every system tick. The capacity should be large enough to process the maximum number of invocations that can occur on system tick timing and it can be estimated in the hyper-period of all the tasks' periods. Every system tick timing, the server processes the task invocations and task switching if necessary. When fewer or no tasks are requested and some of the capacity is left, the remaining capacity is abandoned immediately.

Figure 1 shows an example of intervention of PMS, where there are two periodic tasks, τ_1 and τ_2, and PMS. τ_1 has a period, T_1, of four system ticks and execution time, C_1, of one system tick. τ_2 has six-system-tick period, T_2, and 1-system-tick execution time, C_2. The server has one-system-tick period, T_{PMS}. It is supposed that one system tick corresponds to six processing ticks and that it takes one processing tick to process one task's invocation including task switching. The server capacity, C_{PMS}, of two processing ticks is reserved since two

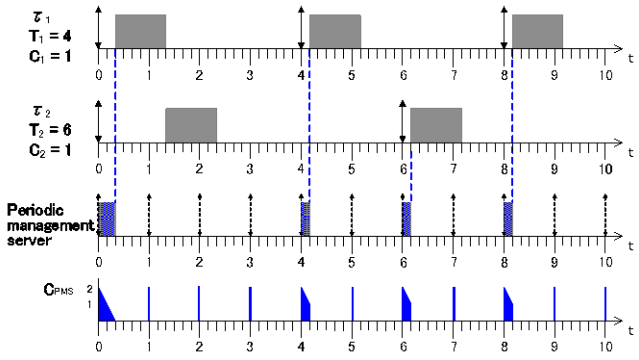


Figure 1: Periodic management server (PMS).

periodic tasks can be requested at the same time in the hyper-period. In the figure, at $t = 0$, the whole capacity is consumed for two tasks' invocations. On the other hand, at t = 1, 2, 3, 5, 7, 9 or 10, no tasks are invoked and the capacity immediately disappears. At t = 4, 6, or 8, a task is invoked and one unit of the capacity is consumed. Then, the remaining capacity is abandoned.

In order to guarantee the schedulability including PMS, the following sufficient condition should be confirmed.
(n is the number of periodic tasks.)

$$\sum_{i=1}^{n} C_i/T_i + C_{PMS}/T_{PMS} \le (n+1)\left[2^{1/(n+1)} - 1\right] \quad (1)$$

This formula is the same as the schedulability test for RM where PMS is regarded as a periodic task[1]. This is satisfied for the example in Figure 1, since, for $C_1 = 1 \times 6$, $T_1 = 4 \times 6$, $C_2 = 1 \times 6$, $T_2 = 6 \times 6$, $C_{PMS} = 2$, $T_{PMS} = 6$, and $n = 2$, the left side becomes 0.750 and the right side 0.780.

3.2 Aperiodic Management Server

The second server, aperiodic management server (AMS), manages aperiodic events such as tasks' completion and task switching. Tasks can finish at any timing different from system ticks. Hence, Polling Server algorithm cannot be used for this server since it discards the capacity if there are not RTOS jobs pending at the system tick timing. Instead, the Deferrable Server algorithm is used as AMS since it preserves the capacity during the system tick period.

In the task model in this study, WCETs are regarded as supposed execution times and the WCETs are estimated as the integral multiple of a system tick. We try to guarantee the schedulability assuming the WCETs. From this assumption, at most one task would finish in a system tick. This leads to that one processing tick for the server capacity is enough to guarantee the schedulability. In actual situation, however, if a task is going to finish before it has spent its WCET, the capacity may be already exhausted. This happens when two or more tasks finish in the same system tick due to their shorter execution times than their WCETs. In this case, blocking the system until the next system tick timing does not jeopardize the schedulability. However, this is a waste of the system resources. Therefore, it is effective to provide two or more processing ticks as the capacity to improve the system utilization.

An example of AMS is depicted in Figure 2. Periodic tasks are the same as those in Figure 1. The server has its period, T_{AMS}, of one system tick (= six processing ticks) and capacity, C_{AMS}, of one processing tick. At t = 1, the server performs τ_1's completion and switching to τ_2. Similarly, whenever a periodic task finishes, the server processes the task completion while consuming the capacity. Note that the server can cope with task's completion even when the task finishes asynchronously with system ticks.

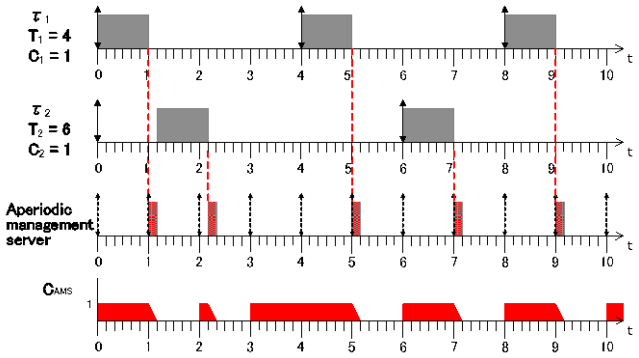


Figure 2: A periodic management server (AMS)

The following sufficient condition should be confirmed to guarantee the system schedulability involving overhead for aperiodic management. (This corresponds to a sufficient condition for Deferrable Server [2] and n is the number of periodic tasks.)

$$\sum_{i=1}^{n} C_i/T_i \le n\left[\left(\frac{C_{AMS}/T_{AMS} + 2}{2C_{AMS}/T_{AMS} + 1}\right)^{1/n} - 1\right] \quad (2)$$

For the example in Figure 2, this condition is met where the left becomes 0.417 and the right 0.550.

3.3 Merging Two Servers

Figure 3 shows a schedule example of a system including both PMS and AMS. Whenever periodic tasks are released, PMS is scheduled and the corresponding amount of capacity is consumed, while AMS is scheduled immediately after each periodic instance finishes and the capacity is consumed. Both of the servers are replenished with their capacity every system tick.

Guaranteeing the schedulability for a system with PMS and AMS requires the following formula 3 to be satisfied. This is derived from the schedulability condition formula 2 in Section 3.2. Developers can statically check schedulability of the systems by using this formula.

[1] When the Polling Server has the highest priority, the condition is expressed as follows [2].

$$\sum_{i=1}^{n} C_i/T_i \le n\left[\left(\frac{2}{C_{PMS}/T_{PMS} + 1}\right)^{1/n} - 1\right]$$

This is satisfied for the example in the figure since the left side becomes 0.417 which is smaller than the right side, 0.449. This formula cannot be directly applied when another type of server, Deferrable Server, coexists in the system. Therefore, the application of this formula is left as future work.

$$\sum_{i=1}^{n} C_i/T_i + C_{PMS}/T_{PMS}$$

$$\leq (n+1)\left[\left(\frac{C_{AMS}/T_{AMS}+2}{2C_{AMS}/T_{AMS}+1}\right)^{1/(n+1)} - 1\right] \quad (3)$$

It can be confirmed that the example in Figure 3 satisfies this condition.

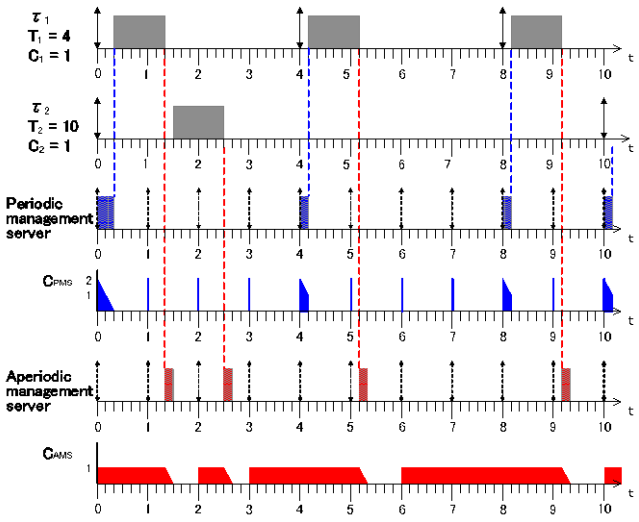


Figure 3: System with PMS and AMS

3.4 Implementation Issues

Generally, RTOSs perform task management and scheduling by employing queue structure which links task control blocks (TCBs). Since the proposed servers are included in the targets of scheduling, it would generate additional space and computational overhead if the control blocks for the servers are prepared and dynamically managed.

Basically if the servers exist only for guaranteeing schedulability for periodic hard tasks, control blocks for the servers are not indispensable or RTOSs do not have to do anything for the server management at run-time. If the slack reclaiming technique described in the next section is applied, the RTOSs have only to manage the server capacities, without their control blocks, in their routines every time they process and block the system until the next replenishment when the capacity is exhausted.

4 Slack Reclaiming

The RTOS servers proposed in the previous section are apt to generate slack, which means the capacity being left unused, due to infrequent RTOS processing. For example, in Figure 3, the total capacity provided to the two servers is 30 between t = 0 and t = 10. However, the amount actually consumed is only 8, which is less than 30% of the total. In this section, techniques to avoid wasting the unused capacity and improve response times of aperiodic application tasks by utilizing the slack are proposed.

In complicated embedded systems, there are not only periodic control tasks but also aperiodic application tasks which have soft or non-real-time requirements. Therefore, aperiodic tasks are supposed to be included in the task set model in this section. Aperiodic tasks are assumed to be able to arrive at processing tick timing. Their invocation and completion are regarded as lightweight processing so that their WCETs include the processing in advance. The aperiodic tasks are served basically in the background. (Although they can be served by more sophisticated fixed-priority servers, the background server is assumed for simplicity.) In the following subsections, how each management server utilizes slack for aperiodic application tasks is described.

4.1 Slack Reclaiming from Periodic Management Server

PMS behaves as a Polling Server. Therefore, it replenishes its capacity every system tick timing. If the number of tasks to be invoked is less than the capacity, the unnecessary capacity is discarded immediately. Instead of discarding it, if there is a pending aperiodic application task, the capacity to be discarded can be given to the execution of the aperiodic task. This is a basic technique of slack reclaiming. Note that the aperiodic execution is performed preferentially since the server capacity has the highest priority, which shortens the response time of the aperiodic request.

Figure 4 shows an example of the slack reclaiming by PMS. In the figure, an aperiodic application request with supposed execution time of one system tick (= six processing ticks) occurs at t = 4. In the top figure without the slack reclaiming, the instances of τ_1 and τ_2 are given priority over the aperiodic task and therefore the aperiodic task's execution is delayed or preempted. As a result, the response time of the aperiodic execution becomes more than three system ticks (20 processing ticks). On the other hand, in the bottom figure which includes the slack reclaiming technique, the response time is decreased too shorter than three system ticks (13 processing ticks) by making use of the remaining server capacity at the highest priority. For example, at t =4, it is known that τ_2 is not invoked until t =6. Therefore, one unit of the capacity can be spent for the aperiodic task's execution. Similarly, at t = 5, the whole two units of the capacity can be given to the aperiodic task's execution since neither τ_1 nor τ_2 is invoked. In addition, at t = 6, the aperiodic task is executed in one processing tick by reclaiming the slack which is not used for τ_1's invocation. If the actual execution time of the aperiodic task is shorter than the supposed time (six processing ticks), the effect of the slack reclaiming can be larger. For example, if the execution finished in one processing tick, the response time with the slack reclaiming would be only one processing tick which is shorter than eight processing ticks in the case without the slack reclaiming.

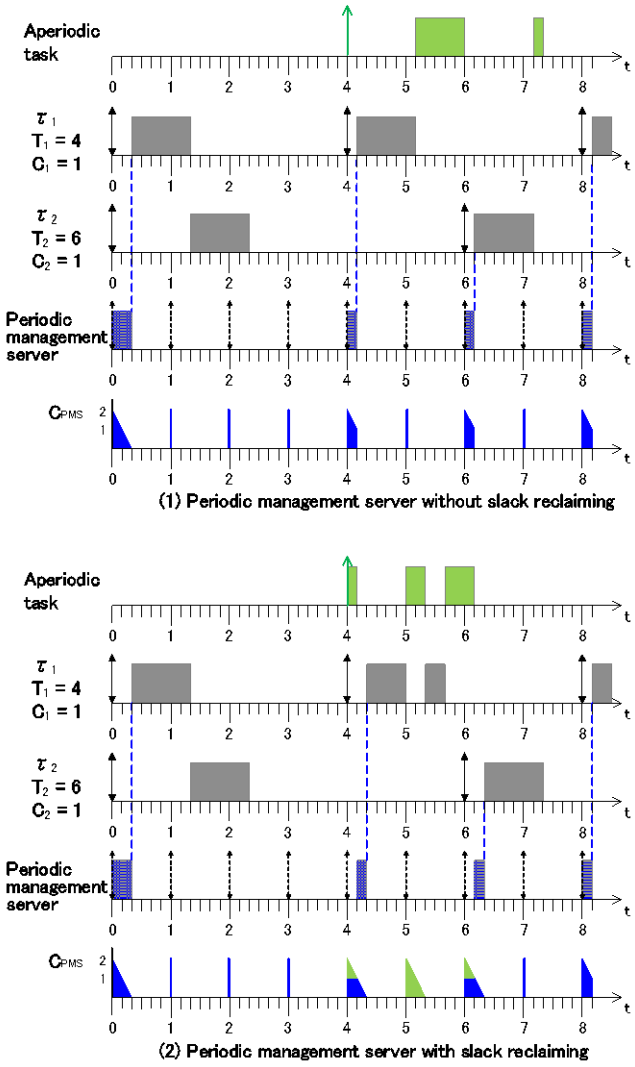


Figure 4: Slack reclaiming from PMS

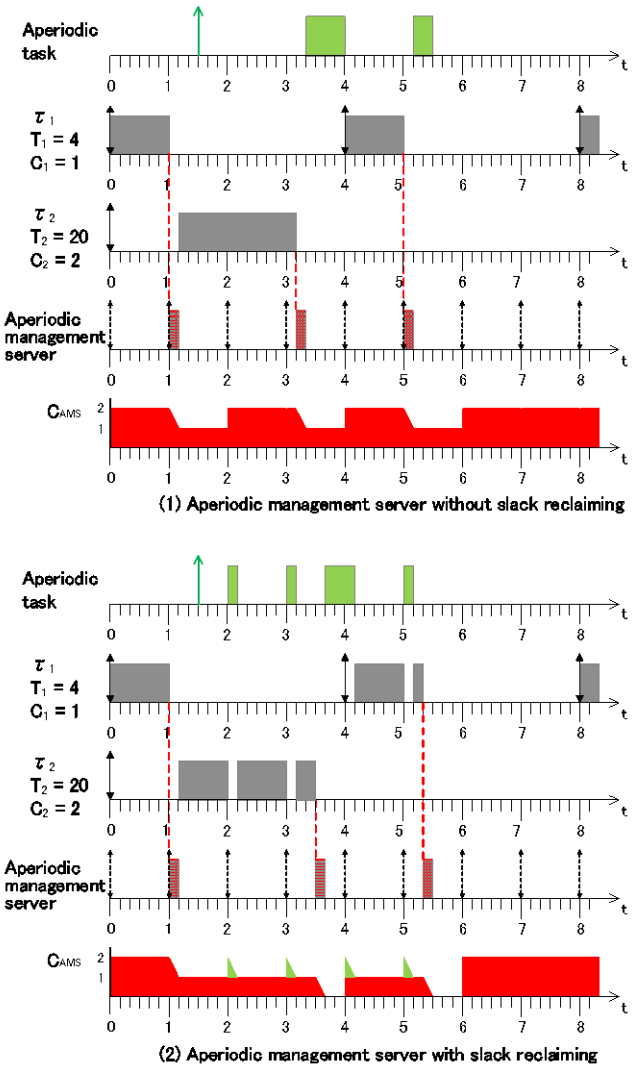


Figure 5: Slack reclaiming from AMS

4.2 Slack Reclaiming from Aperiodic Management Server

Every system tick, AMS has to have the capacity of more than one processing tick for, at least, one task to finish. Two or more tasks might finish in a system tick since task execution can finish earlier than its WCET. Therefore, it is effective for AMS to have the capacity corresponding to two or more processing ticks. In such a case, some capacity might be left unused in a system-tick period since tasks which have already finished and are not invoked until the next period never finish during the system-tick period. The remaining capacity can be utilized by execution of aperiodic application tasks.

An example is shown in Figure 5, where AMS has the capacity of two processing ticks every system tick. There is an aperiodic request with supposed execution time of one system tick (= six processing ticks) arriving three processing ticks after $t = 1$. In the top figure for no slack reclaiming, the instances of τ_1 and τ_2 are given priority over the aperiodic request. The aperiodic task is executed in the background of the periodic tasks and the response time becomes four system ticks (24 processing ticks).

In the bottom figure with slack reclaiming, after τ_1's first instance finishes, τ_1 never finishes until $t = 4$. Therefore, at $t = 2$ and $t = 3$, one unit of the capacity can be provided to aperiodic execution. Similarly, after τ_2's instance finishes, τ_2 does not finish again until $t = 20$. Then, at $t = 4$ and $t = 5$, the corresponding amount of the capacity can be used for the aperiodic execution. As a result, the aperiodic response time becomes three system ticks plus four processing ticks (22 processing ticks). As is the case for the slack reclaiming by PMS, the effectiveness gets larger if the aperiodic task finishes earlier than the supposed finishing time. For example, if the execution finished in one processing tick, the response time with the slack reclaiming would be only four processing ticks while that without reclaiming would be 12 processing ticks.

5 Evaluation

In this section, the proposed slack reclaiming techniques by the management servers are evaluated. The evaluation targets synthetic task sets that are generated using probabilistic distribution. Periods and worst-case execution times of periodic tasks are decided from exponential distribution with the average of 100 and 30 system ticks, respectively. Actual execution times in processing ticks are decided from exponential distribution with the average of 25 system ticks and the upper limit of the corresponding worst-case execution times. We prepared 100 periodic task sets for each of utilization (U_p) between 40% and 70% at intervals of 5%. All the task sets conform to the formula 3, that is, all periodic task executions meet their deadline requirements. In addition, we prepared 10 aperiodic task sets that follow Poison arrival and exponential service times. The utilization by the aperiodic tasks is about 1%. We performed 1,000 (= 100 periodic task sets × 10 aperiodic task sets) simulations with a system tick equal to 100, 50, or 25 processing ticks. PMS and AMS have the same units for capacity as the number of periodic tasks in each task set. The observation period is 100,000 system ticks.

Figure 6, 7, and 8 show average response times of aperiodic application tasks when a system tick equals to 100, 50, and 25 processing ticks, respectively. "None" corresponds to the results without slack reclaiming. "PMS" and "AMS" use the slack reclaiming by PMS and AMS, respectively. "Both" means the slack reclaiming by both the servers. The vertical axis is response times normalized to the results of "None." The horizontal axis is U_p.

In Figure 6, aperiodic responsiveness is improved by up to 3.0% when U_p = 55%. In Figure 7, the improvement becomes up to 2.9% when U_p = 55%. Similarly, the improvement is up to 6.0% when U_p = 55% in Figure 8. It is confirmed that the effectiveness of the slack reclaiming is larger when a system tick is shorter for high degree of precision.

Next, we change periods, worst-case execution times, and actual execution times of periodic tasks from 100 to 300, from 30 to 90, and from 25 to 75, respectively, and investigate how execution times of periodic tasks affects the response times of aperiodic application tasks. Figure 9, 10, and 11 show the results when a system tick equals to 100, 50, and 25 processing ticks, respectively. It can be confirmed that, compared to Figure 6, 7, and 8, effectiveness of the slack reclaiming increases. In Figure 9, the average response time is improved by up to 12.6% when U_p = 55%. In Figure 10, it becomes up to 14.5% when U_p = 60%. In Figure 11, the improvement is up to 18.3% when U_p = 45%. From these results, it can be said that relatively long execution times of periodic tasks lead to higher effectiveness of the slack reclaiming since blocking time of aperiodic requests is mitigated by using unused, high-priority capacity.

The effectiveness of slack reclaiming tends to decline as U_p becomes large, for example, with U_p = 70%. This is because the number of periodic tasks in each task set is not large when

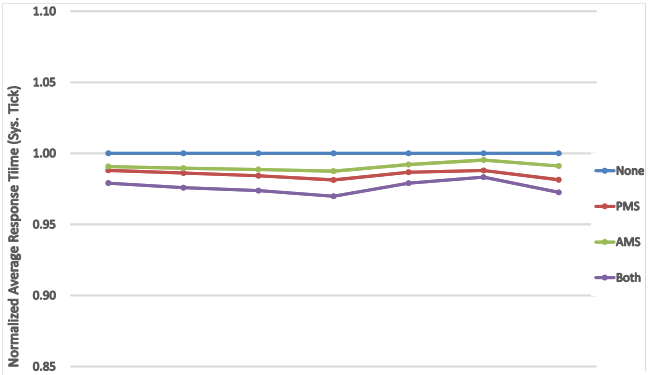


Figure 6: Average response time of aperiodic tasks (1 sys tick = 100 proc ticks, average period = 100 sys ticks, and average WCET = 30 sys ticks)

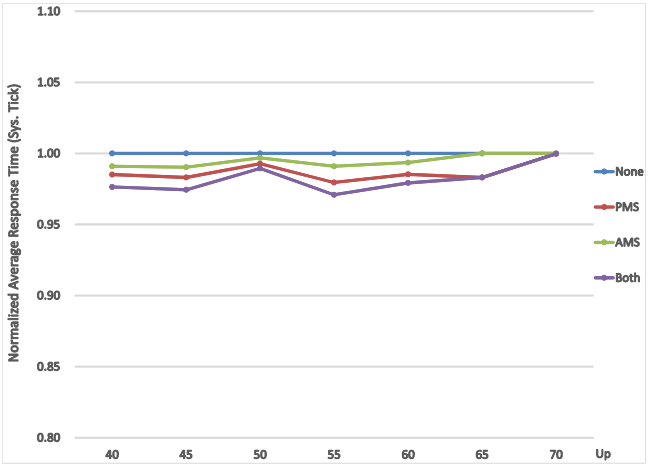


Figure 7: Average response time of aperiodic tasks (1 sys tick = 50 proc ticks, average period = 100 sys ticks, and average WCET = 30 sys ticks)

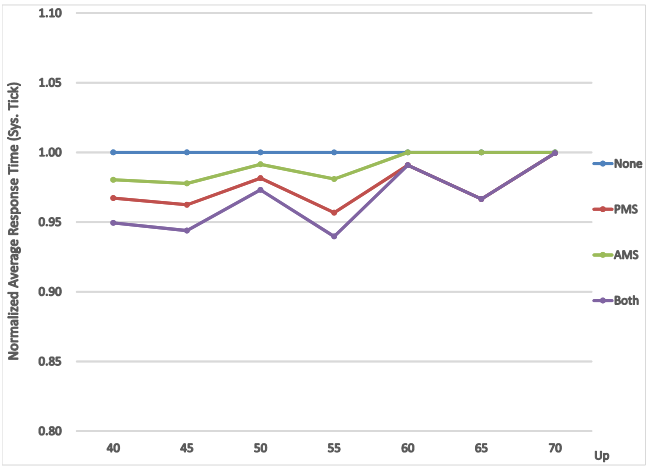


Figure 8: Average response time of aperiodic tasks (1 sys tick = 25 proc ticks, average period = 100 sys ticks, and average WCET = 30 sys ticks)

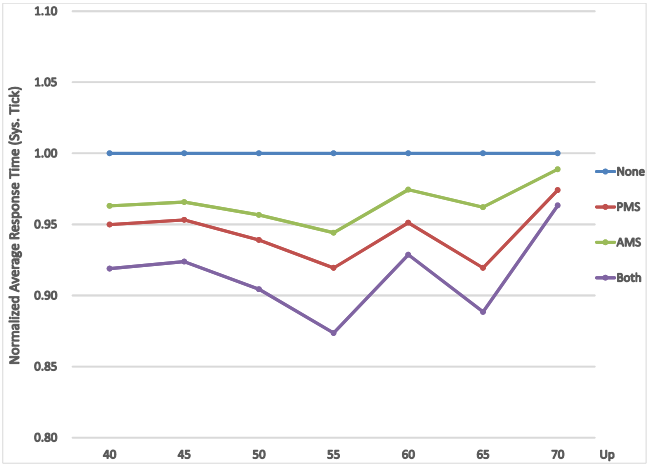


Figure 9: Average response time of aperiodic tasks (1 sys tick = 100 proc ticks, average period = 300 sys ticks, and average WCET = 90 sys ticks)

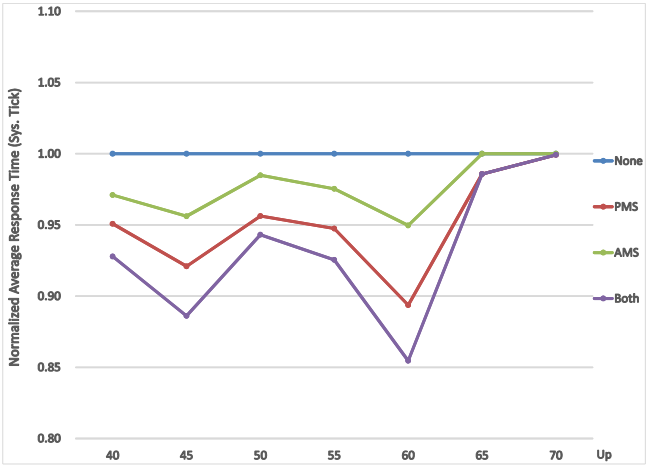


Figure 10: Average response time of aperiodic tasks (1 sys tick = 50 proc ticks, average period = 300 sys ticks, and average WCET = 90 sys ticks).

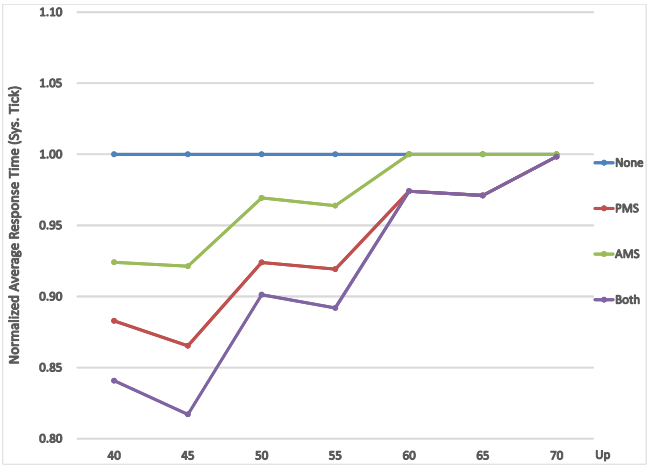


Figure 11: Average response time of aperiodic tasks (1 sys tick = 25 proc ticks, average period = 300 sys ticks, and average WCET = 90 sys ticks).

U_p is high in order to satisfy the formula 3 for schedulability. Table 1 shows the average number of periodic tasks in all the task set. The small number of tasks leads to low server capacity and small effect of slack reclaiming. For example, when U_p is 40%, the servers have the capacity of four units on average, while they have only about 1.8 units when U_p is 70%. Therefore, the amount of reclaiming of slack is low when U_p is 70%.

Table 1: Average number of periodic tasks in all task sets

U_p	Average period	
	100	300
40%	4.00	4.03
45%	3.93	3.90
50%	3.17	3.09
55%	2.97	3.10
60%	2.44	2.52
65%	1.84	1.72
70%	1.75	1.81

6 Conclusions

Use of RTOS is becoming common in embedded systems development to improve real-time processing and shorten the development period. In this paper, considering that run-time overhead of RTOS influences system schedulability, we proposed techniques to assure schedulability including RTOS processing. The proposed techniques include two high-priority servers, a polling server for periodic management such as task invocations and a deferrable server for aperiodic management such as task completions, which are in charge of RTOS processing. It is possible to guarantee schedulability by performing schedulability test in Section 3.3 off-line.

The proposed servers tend to have unused capacity in system periods when periodic tasks do not arrive or finish. To avoid wasting the unused capacity, we proposed slack reclaiming techniques for improving responsiveness of aperiodic application tasks. The techniques provide pending aperiodic tasks the amount of capacity which is not used considering periodicity of periodic tasks, while keeping schedulability of periodic tasks.

Evaluation by simulation with synthetic task sets showed that the proposed slack reclaiming techniques improved aperiodic response times by up to 18.3%. It is confirmed that the effectiveness becomes larger when the system tick is relatively short or when execution times of periodic tasks are relatively long.

In the evaluation in this paper, aperiodic application tasks are executed basically in the background of the periodic tasks. In the future, for more realistic systems, we plan to introduce a server algorithm which takes charge of aperiodic application tasks and evaluate the system.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP 15K00073.

References

[1] S. Baruah, H. Li, and L. Stougie, "Towards the Design of Certifiable Mixed-Criticality Systems," *Proc. of IEEE Real-Time and Embedded Technology and Application Symposium*, pp. 13-22, April 2010.

[2] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd Edition, Springer, 2011.

[3] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the Scheduling of Mixed-Criticality Real-Time Task Sets," *Proc. of IEEE Real-Time Systems Symposium*, pp. 291-300, December 2009.

[4] K. Jeffay and D. L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems*," Proc. of IEEE Real-Time Systems Symposium*, pp. 212-221, December 1993.

[5] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proc. of IEEE Real-Time Systems Symposium*, pp. 261-270, December 1987.

[6] J. P. Lehoczky and S. Ramos-Thue, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proc. of IEEE Real-Time Systems Symposium*, pp.110-123, December 1992.

[7] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the Association for Computing Machinery*, 20(1):46-61, January 1973.

[8] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems," *Journal of Real-Time Systems*, 1(1):27-60, 1989.

Kazuki Hasegawa received B.S. from Tokai University in 2013 and M.S. from Japan Advanced Institute of Science and Technology (JAIST) in 2015. His research interests are high-performance computer architecture, real-time task scheduling theories, operating systems, and embedded systems.

Kiyofumi Tanaka is an Associate Professor at Japan Advanced Institute of Science and Technology (JAIST). He received B.S, M.S., and Ph.D. from the University of Tokyo in 1995, 1997, and 2000, respectively. His research interests are computer architecture and real-time embedded systems. He is a member of IEEE, ACM, IEICE, and IPSJ.

A Unified Cloud Metering Framework

Karim Sobh[*‡§] and Amr El-Kadi[†‡§]
American University in Cairo, Cairo, EGYPT 11835.

Abstract

Cloud computing utilizes the integration of different computing technologies to achieve a utility computing model. Computing resources are consolidated and shared among different applications transparently. Cloud environments are like a market place, and an accurate cloud metering framework is needed for fair chargeback and accurate responsive Service Level Agreement (SLA) policies. A data modeling approach coupled with a scalable distributed architecture is adopted to build a unified cloud metering framework that is based on a Cloud Metering Markup Language (CMML) and a set of network transport specifications. A full prototype of the CMML interpreter is implemented, as well as a Distributed Proc Filesystem as a communication protocol. The statistical methods Analysis of Variance (ANOVA) and Generalized Linear Models (GLM) were used for evaluation purposes and a comparative study between the Distributed Proc File System and plain TCP is conducted to asses the probe effect. A framework end-to-end factorial design experiment based on ANOVA/GLM is conducted to study the different factors affecting the framework behavior. Finally, the results of a case study is presented to demonstrate how the framework implementation performs in a realistic environment, focusing on the characterization of the associated overhead and demonstrating its low probe effect.

Key Words: Cloud metering, cloud computing, metering framework, cloud metering markup language, autonomous cloud metering objects, proc filesystem, kernel level transport layer, netfilter hooks.

1 Introduction

Cloud environments are like a market place. A single distributed application owned by a single user can share different resources owned by different service providers. A cloud application can utilize different resources at different architectural layers, namely hardware, virtualization, and application layer. A resource can be primitive or composite, e.g., a virtual machine is a composite resource that is built up of a number of primitive resources such as CPUs, RAM, virtual disks, etc.

Cloud users are being charged for their usage based on flat rate time plans. The market competition urges for accurate metering standards for charging users. A cloud resource is shared through multiplexing mechanisms, and the framework we have introduced is capable of correlating resources usage across different architectural layers, where by different metering abstraction levels can be achieved.

Our framework is based on a data modeling extensible Cloud Metering Markup Language (CMML) coupled with a scalable multi-tier architecture. Metering Model shareablity, as well as low overhead execution of metering engines running on the cloud resources are key features.

In Section 2 we present the background followed by related work in Section 3. We present the problem characteristics in Section 4 and introduce our framework in Section 5. In Section 6, we present the prototype details of the proposed metering framework. A case study showing the deployment of the proposed framework in a realistic environment is presented in Section 7. Experiment results are presented in Section 8 and in Section 9, we conclude and present our possible future work.

2 Background

Cloud environments consolidate computing resources located in different architectural layers as shown in Figure 1. The complexity of accurate metering arises from multiplexing cloud resources among different applications. Virtualization is another dimension of complexity resulting from unsynchronized virtual clocks, leading to inaccurate metering results from within a virtual machine. Correlating metering data generated from distributed virtual resources is a complex challenging task by nature.

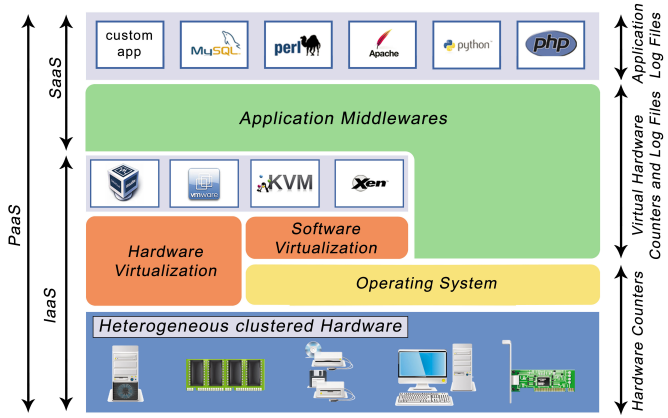


Figure 1: Cloud architectural layers

Figure 2 shows a three-tier metering architecture. Log collection takes place in the front-tier, where metering data incompatibility is exhibited and the need for unification arises.

[*]E-mail: kmsobh@aucegypt.edu
[†]E-mail:elkadi@aucegypt.edu.
[‡]The Department of Computer Science and Engineering
[§]US Patent Application no. 15/088,476. Date: April 1, 2016.

Metering data collected from different sources are correlated in the middle-tier. Metering data storage, billing, and Service Level Agreement (SLA) monitoring are considered back-end metering services. Interaction with the target cloud management middleware is essential for the metering engines to be able to retrieve vital information about the cloud resources to be metered.

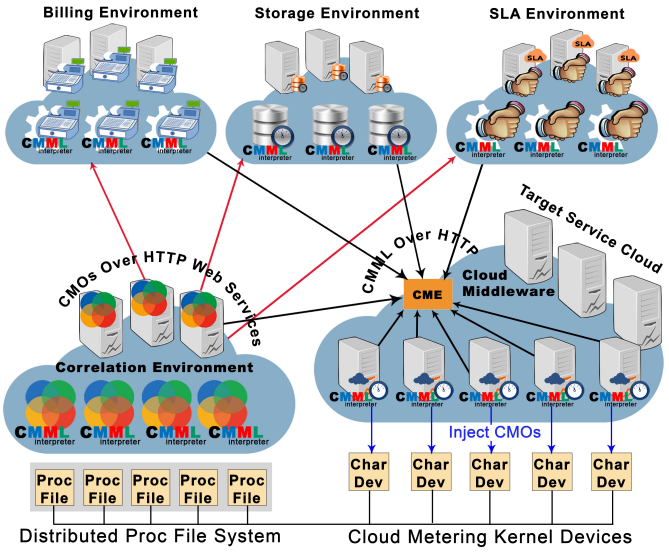


Figure 2: Multi-tier metering architecture

The metering process phases are log collection, unification, transportation, correlation, and back-end processing. Collection engines running on cloud service nodes extract and parse logs, and hence pose as the main source for probe effect. Data transport between collection engines and the correlation tier need to be optimized to reduce the probe effect on the cloud network resources.

3 Related Work

Cloud metering is a new research domain and consequently limited work exists in the literature that tackles the cloud metering problem in a unified approach. A comparative study on different metering domains in distributed systems and cloud computing was conducted, namely power and resource usage, virtual resource usage, log management, billing and accounting, and other attempts of unified cloud metering approaches. A selective representative sample of the related work in each domain is presented in this section.

Power and resource usage, being one of the main sources of raw metering data, is a very important aspect in cloud metering. Aman Kansal et al. presented the Joulemeter in [17] to overcome the lack of power metering within a virtual machine. T. Singh and P.K. Vara presented guidelines for smart metering cloud environments in [30]. A comprehensive study for power consumption in data centers by Anton Beloglazov et al. is presented in [4]. A Digital Continuous Profiling Infrastructure (DCPI) is presented by Jennifer M. Anderson et al. in [1]. Google-wide profiling (GWP), presented by Gang Ren et al. in [27] is a distributed profiler for data centers and cloud environments.

Virtualized resource metering is a very important metering aspect due to the metering difficulties resulting from virtualized resources multiplexing over physical ones. Exposing hardware counters for profiling in virtualized environments is discussed by Benjamin Serebrin and Daniel Hecht in [28]. Jiaqing Du et al. tackled the problem of interrupt forwarding and enabling access to the Performance Monitoring Unit to the guest environment in [6], and an implementation of virtualized profiling on KVM is presented. A metering technique for Virtual Machines based on the Virtual Platform Architecture is proposed in [14] to run from within virtual machines.

Log management is an important building block in any metering process, and a lot of complexities and challenges are entailed in such a task especially in distributed systems. D. Huemer and A.M. Tjoa introduced a solution for log incomparability in [13] through automatic log evaluation based on XML. Predictive Modelling Markup Language (PMML) is presented by Guazzelli et al. in [10] as an open standard for sharing models through coupling data with its operation definitions. The scalable Run Time Correlation Engine (RTCE) is introduced by Miao Wang et al. in [33] for correlating distributed logs, and using dispatchers for load balancing and scalability. William M. Jones et al. presented analytical and simulation-based approaches in [16] showing the negligible impact of choosing a sub-optimal checkpoint. The issues of continuous sampling are raised by Gang Ren et al. in [27]. Jennifer M. Andersonet al. presented in [1] a technique for hardware counters continuous sampling through hardware support on Digital ALPHA systems.

Accounting and billing are very important examples of applications that depend on metering, and would exist in any utility based computing environment. A series of work presented by Francisco Airton Pereira da Silva et al. in [23, 24, 29] for a cloud accounting system and charging policy based on a domain specific language (DSL). The DGAS, Distributed Grid Accounting System, is presented in [25] as an accounting infrastructure for grid environments. The GridBank (GB) is presented in [3] as a secure grid-wide accounting infrastructure service. Ali Anwar et al. presented in [2] a Cost-Aware cloud metering for dynamic revenue scaling, which is concerned with estimating the metering data size for efficient cloud resource scaling.

In [22], Naik, V. K. et al. presented an end-to-end metering framework for federated hybrid cloud services. The presented framework is claimed to solve numerous problems in cloud metering such as single subscription, metering composition over multiple service providers, licenses usage restrictions, integration with legacy accounting and billing systems, and horizontal distribution of workload for better economic resource utilization. Architectural approaches were adopted with less emphasis on the data representation to tackle traditional basic cloud services metering.

4 Problem Definition

The complexity of consolidating cloud resource pools reflects on cloud metering. A resource can be a physical one such as CPU, RAM, disk, etc. or complex virtual resources built on top of more primitive resources; a resource can be a whole environment built up of virtual resources, such as virtual machines, networks, disks, etc. The target metering framework being sought should be able to provide metering perspectives at different levels of abstractions.

Normalization challenges, of hybrid metering data formats, increases with larger resource pools. The ability to correlate different resources usage with their different distributed running cloud applications is an even more insisting problem. Moreover, ability to collect metering data from cloud resources in a seamless and low overhead manner is another dimension of the problem, as it might affect the quality of the running cloud services, and thus exhibiting high probe effect.

We have identified a set of features and design goals that we wanted our target cloud metering framework to exhibit, these are as follows:

(1) **Extensible Representation:** Ease of interpretation and shareability between federated clouds.
(2) **Autonomous Metering Data:** Coupling metering data with their corresponding operations.
(3) **Correlation Capabilities:** Correlation of metering data extracted from different architectural layers.
(4) **Programmability:** Flexibility of defining metering constructs through writing code.
(5) **Standard Metering Transport:** Transporting metering data over simple standard APIs.
(6) **Elastic Multi-tier Architecture:** Can scale with the metering needs.
(7) **Metering Services Redundancy:** For fault tolerance and recovery.
(8) **Low Probe Effect:** Low metering probe overhead.
(9) **Online Metering:** Fast and responsive metering data processing.
(10) **Ease of Integration:** Ease of integration with different cloud environments irrespective of their type, topology, underlying technologies, and service nature.

Based on the above characteristics, our research question can be formulated as "**Does a unified cloud metering framework that can provide extensible, scalable, programmable, and low overhead cloud metering exist? Would the above mentioned characteristics lead to a cloud metering framework that can cope with cloud environment complexities resulting from cloud resources heterogeneity, their existence and execution in different cloud architectural layers?** "

5 The Metering Framework

5.1 Framework Overall View

The metering framework is based on an extensible metering markup data modeling language coupled with a multi-tier scalable architecture. Our target is not a cloud metering system, rather a set of specifications that could be taken as guidelines and/or standards for building different cloud metering systems fitting various target cloud environments.

The extensible object oriented Cloud Metering Markup Language (CMML) is proposed to represent metering data across the framework, through which the concept of autonomous Cloud Metering Objects (CMOs) can be realized. The adopted object oriented model was superimposed over an extensible markup data representation for maximum shareability. Metering data, represented by OO class data members, are coupled with their operations represented by OO class methods. The OO model is further extended with built-in receptors encapsulating routing information within the CMO to enable it to navigate between different framework engines autonomously using self-contained information. The concept of CMOs eliminated the usage of passive metering data through operation definition annotations.

A three-tier architecture was adopted, where each tier can be decomposed further based on the the target functionality of metering. Figure 3 gives an overview of the overall metering framework architecture together with the main metering engines. The cloud environment is considered the metering framework front-end where the metering collection engines are deployed close to their target resources. Collection engines collect raw metering traces and convert them to collection CMOs. Correlation engines are deployed in the middle-tier where collection CMOs are correlated to generate correlation CMOs. The correlation CMOs are sent to the back-end services for further long term processing. All metering engines across the metering architecture should be able to interpret CMOs represented in CMML. Consequently, a CMML interpreter should be deployed to provide a living environment from CMOs.

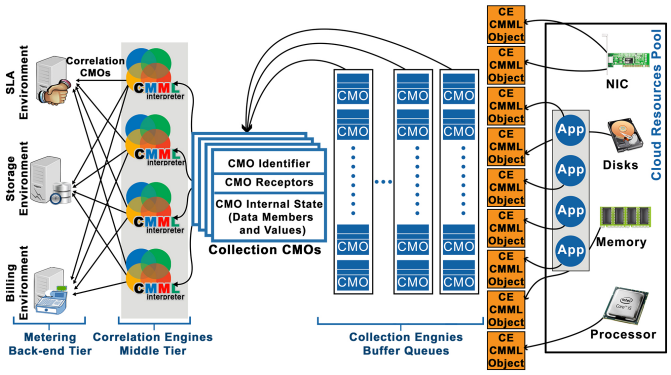


Figure 3: Metering framework architecture

One of the main roles of a cloud middleware is to maintain a resource inventory, and hence a Cloud Metering Extension (CME) is expected to be integrated with the cloud middleware to generate metering CMML scripts, based on resource types and relations, to be downloaded and executed by different metering engines. The (CME) is a core service used by all metering engines as shown in Figure 2.

5.2 Cloud Metering Markup Language (CMML)

CMML is a markup language with functional capabilities. A CMML tag is a construct that executes corresponding logic by a target CMML interpreter. Two mandatory tags need to exist in a CMML script, namely ”CMMLScript” and ”CMMLMain”. The CMMLScript tag encloses the whole script body, and the CMMLMain tag identifies the main entry point for the script execution. A CMML tag can be invoked by name via its ”Name” sub-tag. The ”CMMLRoutine” tag is used to define routines to support modular programming. Concurrency is built at the core of the language. The ”Thread” tag is used to activate tags execution as threads, and can define threads affinity configuration upon needs.

Listing 1 presents a ”Hello World” CMML Script that demonstrates the basic features of the language. This script should print ”Hello World” twice, through invoking the CMMLRoutine and the CMMLOut tag by name. Notice that the two ”Exec” calls will run in parallel as the routine tag has the ”Thread” sub-tag enabled.

```
<CMMLScript>
    <CMMLRoutine>
        <Name>PRINT HELLO WORLD</Name>
        <Thread>TRUE</Thread>
        <CMMLOut>
            <Name>HELLO WORLD</Name>
            <Subject>Hello World !!</Subject>
            <Target>
                <PipeTo>STDOUT</PipeTo>
            </Target>
        </CMMLOut>
    </CMMLRoutine>
    <CMMLMain>
        <Exec>PRINT HELLO WORLD</Exec>
        <Exec>HELLO WORLD</Exec>
    </CMMLMain>
</CMMLScript>
```

Listing 1: CMML hello world script

CMML supports object oriented capabilities as well. Listing 2 shows a simplified CMML class definition for collecting VM CPU data. Each class has a name, set of data members, and set of methods. The CMML object model is extended to support metering constructs. A set of tags are defined in the class definition to hold CMML logic that can execute at different stages of the metering processing, namely ”Collect”, ”Correlate”, ”Bill”, and ”SLA”. Each tag is executed by a metering engine based on the location of the CMO at the time of execution. Each CMML object can be executed as a thread through invoking the built-in predefined implicit method ”start” which invokes the CMML class ”Collect” tag, implicitly.

The CMML Object Model was also extended to a Distributed Object Model based on service state migration. Special CMML built-in serialization tags are sup- ported, namely ”CMMLObjectXMALalize” and ”CMMLObjectCMMLalize”. The adopted mode of operation is that CMOs are suspended and serialized via the ”CMMLObjectXMALalize” tag, as in Listing 2, sent over the network to another metering engine, restarted into the destination CMML runtime environment via ”CMMLObjectCMMLalize”, and resume via the CMML tag corresponding to the destination.

```
<CMMLClass>
    <Name>VMCPUStat</Name>
    <DataMembers>
        <DataMember>
            <Name>VMName</Name>
            <Visibility>PRIVATE</Visibility>
            <Type>string</Type>
            <Exportable>true</Exportable>
        </DataMember>
        <DataMember>
            <Name>cpustat</Name>
            <Visibility>PRIVATE</Visibility>
            <Type>integer</Type>
            <Exportable>true</Exportable>
        </DataMember>
    <DataMembers>
    <Collect>
        <NextCollectionDelay>2</NextCollectionDelay> <!-- Sleep 2 Seconds -->
        <Iterations>0</Iterations> <!-- Runs for ever-->
        .......
        <CMMLObjectXMLalize>
            <CMMLObject>this</CMMLObject>
            <RedirectTo>
                <PipeTo>FILE</PipeTo>
                <PipeName>/dev/CloudMeterDev0</PipeName>
            </RedirectTo>
        </CMMLObjectXMLalize>
    </Collect>
    <Correlate> ..... </Correlate>
    <Billing> ..... </Billing>
    <SLA>.....</SLA>
    <Methods>
        .....
        <Method>
            <Name>GetCPUStats</Name>
            <Body>
                <CMML>.....</CMML>
            </Body>
        </Method>
    </Methods>
</CMMLClass>
```

Listing 2: VMCPUStat class definition

5.3 Transport Layer

The framework specifications mandate that communication between the collection engines and the middle-tier be carried out over standard filesystem I/O operations. Collection engines run on cloud nodes with diversified specifications and capabilities, and a simple as well as primitive data transfer mechanism available on most operating systems is needed. This would provide needed flexibility for the implementation of the transport layer on a range of possibilities (i.e., ranging from a simple file transfer to a sophisticated distributed filesystem.)

A REST/HTTP web service protocol was adopted between the correlation engines and the back-end services, as well as between the framework engines and services deployed outside the framework. This allows for a standardized communication, and decouples the metering services’ execution from the communication operations. The REST protocol is a very primitive web service protocol that provides a lot of implementation flexibility and provides the freedom of superimposing more complex protocols like SOAP, or XML-RPC based on the need.

5.4 Metering Engines

5.4.1 Collection Engines. Collection engines instantiate objects of classes downloaded from the CME and represent resources to be metered. The ”Collect” Tag enclosing the data collection logic is invoked in detached threads. As per Listing 2, the ”NextCollectionDelay” represents the inter-collect-gap in seconds between every execution of the ”Collect” tag body. The ”Iterations” define the number of times the ”Collect” tag body should be executed before the CMML object thread terminates, with zero indicating an endless run. The ”Collect” tag logic

should perform collection, preprocessing, CMO serialization, and injection into the transport layer.

5.4.2 Correlation Engines. CMML classes are downloaded from the CME and instantiated by the correlation server CMML runtime environment. All resource classes are aggregated into wrapper objects that group related resources. The correlation engines read serialized CMOs via filesystem I/O operations. The receptors of each CMO is extracted and the target correlation engine CMML objects are identified. The CMO is then deserialized, started, and passed to the target correlation engine objects as a parameter upon invoking the "Correlate" tag. After correlating all CMOs, the resulting Correlation CMOs are sent to the back-end services over REST/HTTP. The correlation tier can be decomposed into hierarchical sub-tiers where by different processing stages can be defined and established to represent different correlation abstraction layers, and hence different metering perspectives.

Correlation engines perform data and time correlation. Based on the CMOs receptor definitions, related CMOs are grouped and data correlation is achieved. The time correlation is based on the existence of a virtual clock across the framework, and the mechanism for implementing it is left to be decided on at implementation time. The following are two time related correlation mechanisms adopted by the framework specifications.

Adhoc Correlation. CMOs are considered related if they arrive at the correlation engine in the same time frame. This mode of operation is very light weight and does not need intensive computing resources to carry out the needed correlation. This mode should only be used when commutative usage evaluation is needed, or when monitoring specific thresholds of the cloud services usage.

Epoch-Based Correlation. CMOs are timestamped and grouped in time epochs with preconfigured lengths. CMOs belonging to the same time epoch are correlated together and the resulting correlation CMOs are stamped by the start and end timestamps of the epoch. A crucial performance problem is encountered when the rate of collection CMOs is higher than the processing rate. This might hinder the stability and the responsiveness of the correlation environment, and consequently two runtime configurations are constructed to overcome this situation:

(1) **Exact**: The correlation process is terminated if it exceeds the duration of the corresponding epoch. This case can be used if the CMOs represent commutative metering and detailed break down of the metering indicators is not important, e.g. CPU time from the proc filesystem which represents the time of a process since it started.
(2) **Adaptive**: A feedback mechanism between the correlation engines and the CME should be in place for reporting the percentage of CMOs processed post the correlation duration. The CME should automatically change the inter-collect-gaps represented by the "NextCollectionDelay" at runtime to reduce the CMOs generation rate. This process should be performed iteratively until equilibrium is reached.

5.4.3 Storage Engines. The storage engines are back-end services deployed on storage servers. A storage server receives its corresponding storage engine definitions from the CME. The storage servers receive correlation CMOs and store them into corresponding storage engines based on the receptors definition.

5.4.4 Billing Engines. The billing engines are back-end services deployed on billing servers. A billing server receives correlation CMOs based on their receptors and execute the logic enclosed in their "Bill" tag. The billing operations generate billing CMOs that are stored in special billing storage engines.

5.4.5 SLA Engines. The SLA engines are back-end services deployed on SLA servers. An SLA server receives correlation CMOs based on their receptors and execute the logic enclosed in their "SLA" tag, which should perform actions that need to be executed based on usage thresholds that are represented by the CMO data members.

6 Metering Framework Prototype

We developed a fully functional prototype to assess the applicability of the proposed framework. Thus this implementation represents one possible realization of the framework, other realizations are feasible.

6.1 Prototype Components

6.1.1 CMML Interpreter. A CMML interpreter was built on top of an extendable C++ framework. The abstract class CMMLService needs to be inherited by each CMML tag class. The CMMLService encapsulates all the threading and common functionalities needed by a CMML tag, and the CMML tag implementation includes only the business logic. Each CMML tag is compiled into a dynamic shared loadable module which has specific interfaces to be invoked by the interpreter.

6.1.2 Distributed Proc Filesystem. The proposed transport layer is based mainly on extending the UNIX proc filesystem for communication between the front and middle tiers. The transport layer prototype is implemented under LINUX OS, yet the concept adopted can apply to any standard UNIX environment. The transport layer adopts a client/server communication model. Collection engines are at the server side and the correlation nodes are the clients. The cloud service nodes deploy a character device kernel module extension used as a filesystem interface for collection engines to inject their serialized CMOs. The character devices act as a kernel buffer queue for CMOs to be transported over the network.

The cloud metering correlation nodes host the transport layer client side which is a proc filesystem kernel module extension. Two proc directories are created, one for physical nodes and the other for virtual machines. Character devices can register on one or more correlation nodes. Upon registering, the kernel extension creates a proc file entry under one of the two folders based on the type of the node. The proc file entry is named using the node network address.

Upon reading a proc file entry the content of the corresponding character device is transferred over network. Both kernel modules have a netfilter hook activated to handle

the communication which is based on packet reorder and group packets acknowledgement. This allows the correlation nodes to read metering data transparently through standard POSIX filesystem I/O operations. The whole communication transport layer resides inside the kernel space. Overheads are being avoided through intercepting the network packets at network layer 3. Figure 4 illustrates the transport workflow.

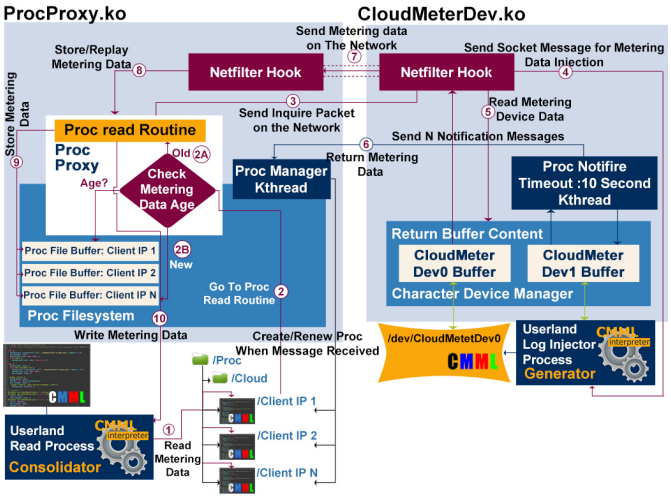


Figure 4: Transport workflow

The Netfilter hooks extension mechanism for the Linux Kernel built-in firewall, allows for adding custom code for packet interception, inspection, and manipulation. Figure 5 shows the five main available hooks that are located at different stages of packet processing for adding custom code. Request and reply packets will be intercepted based on the port numbers in the IN_IP_PRE_ROUTING stage. There are no "userland" processes acquiring the interception ports, and hence considered pseudo ports for packet identification by the netfilter hooks. This will decrease the communication overhead, and will ease the network operation and packets manipulation within the kernel space.

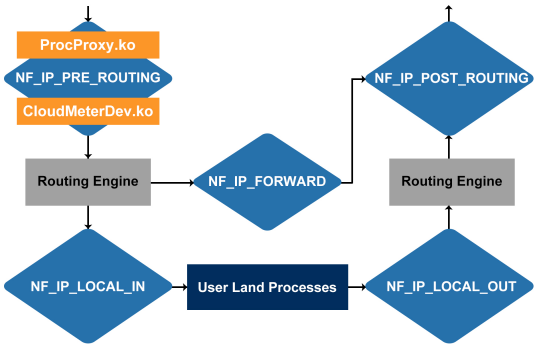


Figure 5: Netfilter hooks

The biggest challenge in using netfilter hooks is that the hooked code is invoked by an interrupt, during which network I/O operations are disallowed. Consequently, kernel work queues are used for deferred I/O task execution. Although this mechanism solves the problem, yet it imposes synchronization complexity where the main thread of execution will not be aware when the I/O operation is finished.

6.1.3 Cloudsparc: Cloud Management Middleware. Cloudsparc is a homegrown IaaS cloud management middleware built on top of an extendable C++ framework. It supports hybride hypervisor technologies, namely QEMU/KVM [19, 26], Oracle VirtualBox [31], and VMWare VMPlayer [32]. Cloudsparc manages a cluster of physical nodes representing the cloud resources pool. Cloudsparc nodes are either master or slave. Master nodes are responsible of managing the cloud configuration, maintaining resources inventory, managing slave nodes, and running cloud services in the form of virtual machines. Slave nodes are responsible only for running cloud services.

A VM template manager is available allowing the creation of VM Profiles with different virtual machine specifications for streamlining the creation and the dispatching of VMs. Cloudsparc is designed to provide control over virtual machine dispatching, ranging from fully automated virtual machine scheduling to the ability of assigning specific virtual machines to specific physical nodes. Cloudsparc allows creating virtual architectures designed for building perspectives of the available resources. A virtual architecture is a grouping mechanism that allows grouping multiple VM profiles and all their dispatched virtual machines can be seen as one processing unit irrespective of their locations.

6.1.4 Cloud Metering Extension (CME). Cloudsparc was extended to support metering through the integration of a CME plugin built up of three main extensions:

Cloudsparc Inventory Extension. The resources inventory is extended for adding definitions of correlation, storage, billing, and SLA servers. Different resource usage price lists and threshold limits can be defined and linked to billing and SLA engines respectively.

Cloudsparc Metering Plugins. The VM Profile is extended to include a metering plugin data structure. A metering plugin is designed to meter a specific cloud resource, e.g. CPU, Memory, I/O, Network usage, or even correlation engines responsible for correlating the collection CMOs generated by the metering plugin, and the storage engines responsible for storing them, if needed. Metering plugins are designed to run on physical nodes as well as from within a virtual machine.

Cloudsparc Correlation Plugins. A correlation plugin defines a correlation engine and the correlation server it should reside on. Storage, Billing, SLA severs, price lists, and SLA thresholds are defined for processing correlation CMOs. Redundant definitions of back-end servers is adopted to achieve redundant metering processing paths, where by the needed infrastructure for fault tolerance is established. Two types of correlation plugins are available, namely metering domes and correlation plugins. Metering domes can group different correlation plugins which provides a second level aggregated

metering perspective. Correlation plugins are designed to correlate metering data coming from virtual machines which belong to either a virtual architecture or a VM profile.

6.2 Metering Framework Workflow

As per the diagram in Figure 2 the CME is a centralized service maintained by the cloud middleware with access to the cloud resources inventory, relations between different resources, and metering configurations to be applied. The CME is invoked by all metering engines upon their startup to download corresponding CMML metering scripts, and to periodically check for updates and changes in metering configurations during their execution. A set of CMML template classes designed to meter different resources are used by the CME to generate CMML scripts on the fly upon metering engine invocations. The CME is also responsible for providing time information via a modified version of Berkeley's Algorithm [11, 12] to establish a distributed common virtual clock.

The collection engine starts with a seeder CMML script as in Listing 3 to connect and download the metering CMML script from the CME. The powerful tag, CMMLRemoteInclude, is used to initiate a REST/HTTP request to the CME with parameters identifying the nature of the request. The CME identifies the invoking collection engine from the network connection parameters, and the "SECONDARY_ACTION" parameter, and prepares the corresponding CMML script based on its inventory and the metering configuration. The generated CMML script is loaded instantly into the interpreter runtime environment. The generated CMML classes are responsible for metering the resources on the corresponding node, either physical or virtual. The downloaded CMML script contains CMML constructs to start the collection engine's character devices loadable module. The character device registers itself on one or more correlation server's distributed proc filesystem, based on the CMML script, for fault tolerance purposes.

```
<CMMLRemoteInclude>
    <Server>[#CME_IP_ADDRESS#]</Server>
    <Port>9999</Port>
    <Method>POST</Method>
    <ServiceName>/FetchMeteringCMMLScript.cgi</ServiceName>
    <Fields>
        <xml_request>
            <Cloud>
                <FetchMeteringCMMLScript>
                    <Action>CUSTOM</Action>
                    <SECONDARY_ACTION>COLLECTION_ENGINE</SECONDARY_ACTION>
                </FetchMeteringCMMLScript>
            </Cloud>
        </xml_request>
        <response_mode>TransactionResponse</response_mode>
        <UNAME>metering</UNAME>
        <PASS>metering</PASS>
        <LOGIN>Login</LOGIN>
    </Fields>
</CMMLRemoteInclude>
```

Listing 3: CMML collection engine seeder

The collection engine instantiates one or more objects from each CMML class to run in detached threads. The CMML objects will continue to inject collected serialized CMOs to a synchronized managed shared buffer through executing the CMML code enclosed in the "Collect" tag. A special shared buffer manager is designed to watch the size of the buffer, and upon reaching a pre-configured size, which we refer to as the queue size, the buffer manager will add Time-To-Live (TTL) information to the buffer content and inject the CMOs into the distributed proc filesystem character device. The TTL information is used by the correlation engines to calculate the time of the next read.

The correlation engines, started via seeder CMML scripts, invoke the CME to retrieve relationship information between different resources, and instantiate aggregate CMML objects accordingly. The correlation engines probe their proc filesystem periodically based on the TTL infor- mation returned in the CMML streams. Different CMML objects are deserialized, restarted, and undergo correlation processing, based on their receptors, and through invoking the "Correlate" tag. The resulting correlation CMOs are submitted to the back-end tier over web services. A typical correlation server, deploying a CMML interpreter, would use the CMML presented in Listing 4 for continuous correlation.

```
<CMMLObjectCorrelate>
    <Name>CorrelationManager</Name>
    <CreateAhead>1</CreateAhead>
    <Subject>
        <CMMLFetchDirContent>
            <Name>metering_proc_fetcher</Name>
            <Directory>/proc/cloud/</Directory>
            <Directory>/proc/vcloud/</Directory>
            <ReadSize>204800</ReadSize>
            <CreateAhead>1</CreateAhead>
            <HuffmanCompressed>true</HuffmanCompressed>
            <LogFile>correlation</LogFile>
            <CacheSize>104857600</CacheSize>
            <SleepInterval>1</SleepInterval>
        </CMMLFetchDirContent>
    </Subject>
    <GracePeriod>2</GracePeriod>
    <Slicing>
        <Duration>30</Duration>
        <Slices>6</Slices>
        <Method>Adaptive</Method>
        <FeedbackIP>192.168.1.20</FeedbackIP>
    </Slicing>
</CMMLObjectCorrelate>
```

Listing 4: CMML correlation engine

The CMMLFetchDirContent is a CMML tag designed to run in a detached thread. Upon first invocation of the tag the thread is created and detached, and continuously monitors the provided filesystem directories in the "Directory" tag for new data, which is stored in the tag's internal buffer. TTL information is utilized for better utilization of the underlying filesystem. On subsequent invocations the tag returns its internal buffer to the invoker and flushes it. The CMMLObjectCorrelat tag is a CMML tag responsible for CMO correlation. The tag is designed to execute as a thread to continuously correlate incoming CMOs into its "Subject" tag. The Subject tag encloses the invocation of the CMMLFetchDirContent to fetch new CMOs. The GracePeriod tag defines a sleep duration in seconds between every correlation attempt. The correlation mode can be configured using the composite "Slicing" tag.

Resulting Correlation CMOs are forwarded to the different back-end services based on their receptor configurations. The storage engines will store the correlation as well as the collection CMOs. The billing engine will execute the "Bill" method represented by the "Bill" tag enclosing the billing logic to generate and store bills. The same applies for the "SLA" tag with the capability to take actions upon SLA violations such as CPU capping and network bandwidth reduction.

The most important thing here is that the model allows for dealing with autonomous CMOs that encapsulate data and operations, rather than collecting metering data and deciding on the metering operations in a later stage. Moreover, the framework is capable of metering cloud resources at various levels of abstractions with ease through the flexibility of writing code; a feature that exhibits the programmability and extensibility extents of our framework.

7 Case Study

We have chosen a real online shop application as a case study to demonstrate the metering capabilities of the framework and the ease of integration with an already existing application. The application was designed and implemented without having cloud deployment or metering in mind. A new requirement is introduced, which is charging the shops' customers for the computing resources they use while they are performing their purchase transactions.

The online shop provides a categorized products menu. An online user needs to register on the website and provide personal details for verification in order to make purchases. A shopping cart engine is integrated into the system allowing users to select their products, add them to their shopping cart, and check out after reviewing the list of products in their shopping cart together with the price details and totals. The application is a real one that was developed 10 years ago and went through a series of upgrades and enhancements to reach its current state.

7.1 Online Shop Environment

The online shop web application is based on open source technology. The application is written in PHP and deployed on an Apache web server. A back-end MySQL database is used to store the online shop data, as well as the Content Management System (CMS) configuration. The application was designed with scalability in mind, where the application can be deployed on a horizontal cascaded Apache web server farm. The back-end database can be configured using MySQL Replication cluster to enhance performance through load distribution.

The online shop was deployed on 5 virtual architectures, each representing a shop branch. Each shop branch environment has 4 virtual machines acting as an Apache web server farm, one virtual machine acting as a web traffic dispatcher to distribute the load over the Apache web servers, and one virtual machine acting as a back-end database. The 5 back-end database machines of the 5 branches are configured using MySQL Master/Master Replication. The diagram in Figure 6 best describes the cloud deployment environment for the case study.

Each virtual machine follows a VM Profile. Four physical metering plugins are defined to meter the virtual machine resources, and another four virtual metering plugins are defined for metering the different services based on the VM Profile type, namely Apache, HAProxy, and MySQL. A metering plugin for metering the usage of the online shop application is defined for the Apache VM profile. The metered resources are CPU, RAM, I/O, and network.

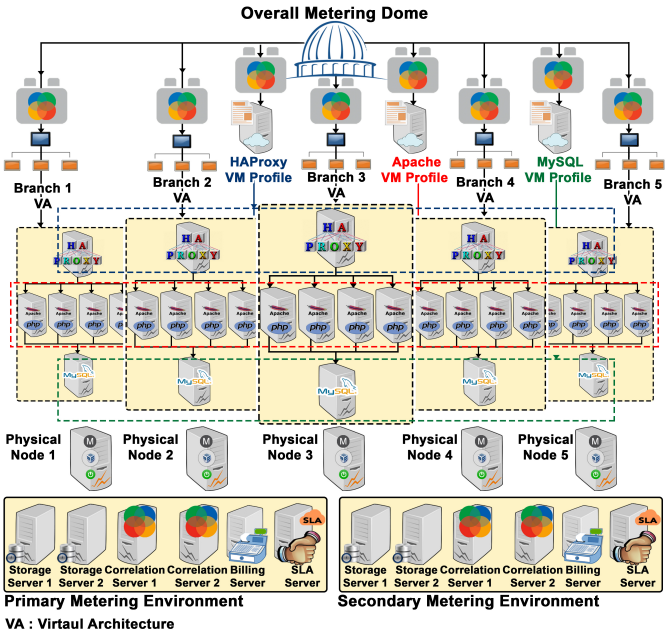


Figure 6: Case study deployment and metering environment

Five correlation plugins are defined to monitor the resources of the five shop branches, by assigning the correlation plugins to the shops' virtual architectures. Another three correlation plugins are defined to monitor the different online shop services by assigning each correlation plugin to a VM Profile. Finally, a **metering dome** is defined to include and aggregate the usage of the 5 shop correlation plugins to provide an overall online shop perspective.

The middle and the back-end tiers of the metering environment are deployed on dedicated virtual machines. The metering engines are distributed over two correlation, two storage, one billing, and one SLA servers. Figure 6 shows the redundant metering environment represented by the primary and the secondary set of resources to establish the prerequisite infrastructure for fault-tolerance.

7.2 Online Shop Metering Extension

Metering data needs to be presented in different perspectives and abstraction levels; namely data center, service provider, and online shop user usage. Consequently, new metering plugins are introduced and limited amendments, to the online shop web application, are carried out.

7.2.1 CMML Metering Plugin. A metering plugin is defined in the Apache VM profile to collect metering data from the online web application as per users transaction. The diagram in Figure 7 shows the details of the Apache VM collection engine. Four virtual metering plugins are deployed in each Apache virtual machine to collect CPU, Memory, I/O, and Network usage. The magnified web application metering plugin acts as a web server receiving usage indicators in white-space delimited format over REST/HTTP. The metering data is parsed and loaded into the internal state data members of the plugin CMML object.

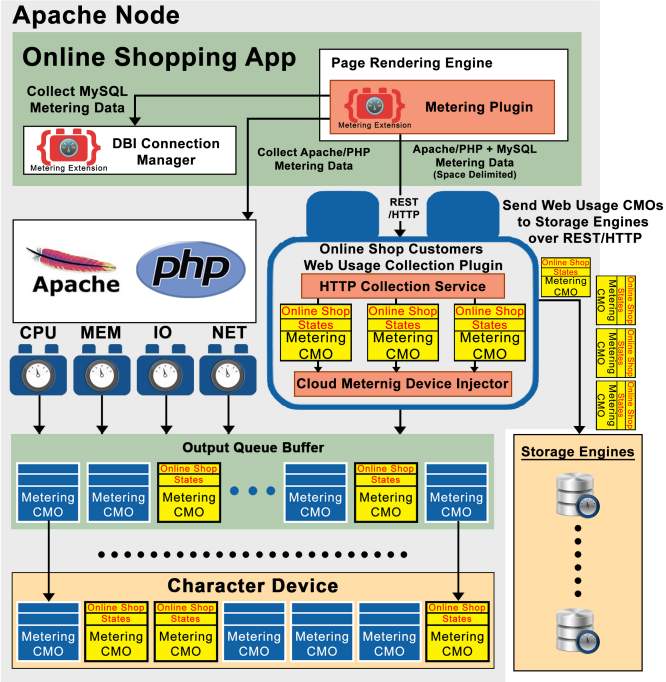


Figure 7: Online shop metering plugin

The web application CMOs are periodically serialized and injected into the collection engine shared buffer to be integrated with the over all metering model. The CMOs are also submitted to a special temporary billing storage engine allowing the shopping cart to present the user bill at checkout within the same session, and avoiding the need to wait for the correlation process to finish.

7.2.2 Web Application Extension. The web application needs to generate usage data and post it to the metering plugin web interface. The diagram in Figure 8 shows the different components of the web application, which are built using a web CMS that supports page templates and a page rendering engine.

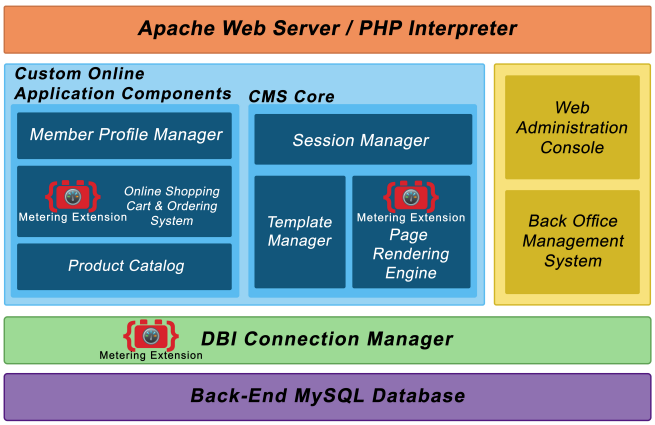


Figure 8: Online shop web application metering extension

Code amendments are applied to the Database Interface Connection Manager (DBI), the page rendering engine, and the shopping cart checkout script. The DBI encapsulates all the database operations and MySQL profiling is enabled within the DBI to collect MySQL usage indicators with respect to CPU, Memory, and I/O. The PHP getrusage command is used in the page rendering engine to collect CPU usage of web transaction executions. The data bandwidth is calculated through measuring the size of the HTTP reply data stream. Finally, metering indicators as well as the corresponding logged in user information are encapsulated in a white-space delimited string and posted to the metering plugin web server through the PHP/CURL library.

An XML/XSL engine is built to invoke the web application billing storage engine from within the shopping cart checkout form. A list of web transactions, together with their usage and billing details, is presented to the user and the cost of the computing resources used in the performed shopping transactions is added to the total cart bill.

7.2.3 Discovery Service and Console. A generic discovery service and console were developed using standard web technologies, mainly XSL/AJAX. It is capable of parsing CMML records and discovering the embedded relation between different usage metering records without prior knowledge of the metered resources. The console is designed to present metering data at different abstraction levels, which acts as a demonstration example of the extensible shareable nature of the CMML representation.

7.3 Functional Demonstration Experiments

A Web Load Generator (WLG) was used to apply different workloads to the case study environment in an experiment designed to showcase the framework functional capabilities, and how it reacts to different workloads and pricing schemes. A comparison between constant workload with variable pricing and variable workload with constant pricing was conducted. The charts in Figure 9 show the results with respect to a unified cloud unit of payment.

Plot (A) shows the different total bills generated per branch for fixed workload and different resource pricing, while plot (B) shows the effect of the different workload on the bills when the pricing is fixed. In plot (A) the overall shop bill was not the maximum among the branches although it represents aggregation of all the branches usage, while on the contrary in plot (B) it represents the aggregate of all bills. The results show the responsiveness of the framework to different configurations and the accuracy in differentiating usage based on workload and price schemes. Plot (C) and (D) compare the branches with respect to the values of the bills generated over time. Plot (E) and (F) show a branch revenue percentage comparison. Finally, plot (G) and (H) show a comparison between member bills based on their transactions. In plot (G) all the members will pay the same since they all have identical transactions across all branches, while plot (H) shows different member payments based on the number of orders they performed.

The charts in Figure 10 show a comparison CPU usage of different branch application services, namely Apache,

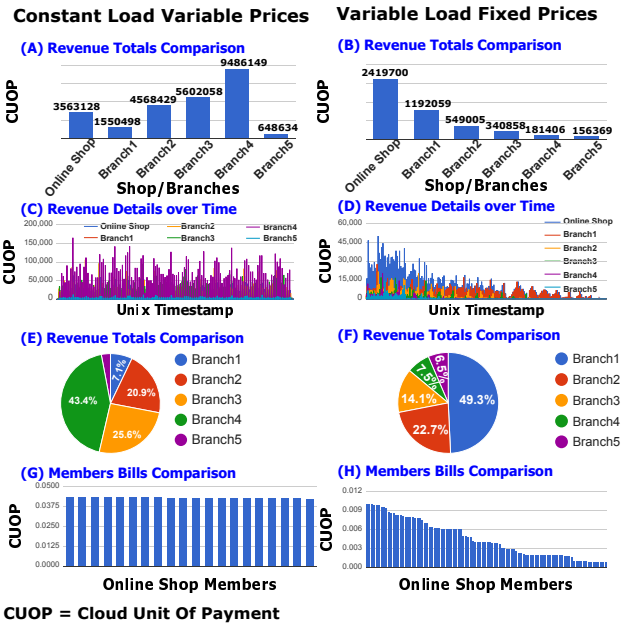


Figure 9: Online shop load/prices comparison

HAProxy, MySQL, and the VM hypervisors. Plots (A), (B), (C), and (D) represent the CPU usage of the mentioned services respectively, showing a relatively uniform CPU usage in the fixed load setup. Plots (E), (F), (G), and (H) show the effect of variable workload applied per branch on the CPU usage of different branches and services.

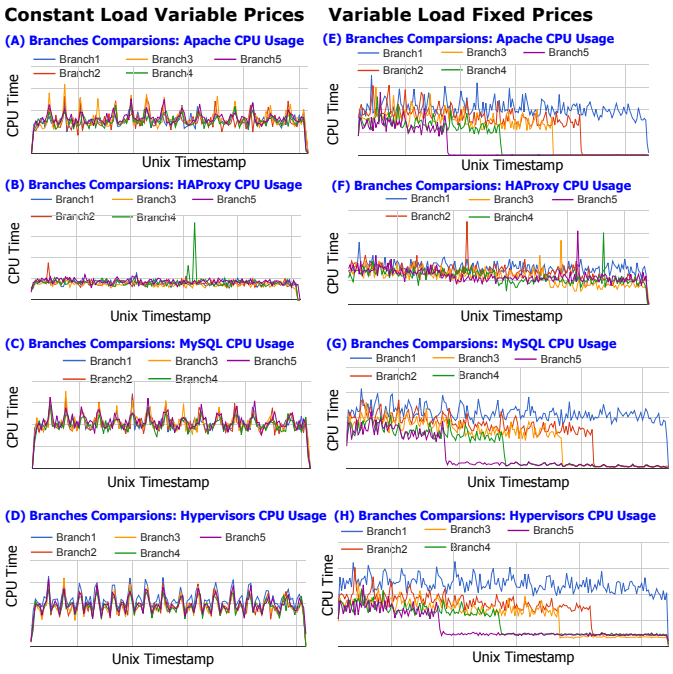


Figure 10: Online shop branches/services comparison

The multi-perspective results presented and generated by the discovery service/console showed the ability of the framework to correlate and aggregate branch resources usage by type of service across all branches. The application usage was integrated into the correlation process and granular bills were issued as per application user over pre-configured time slices. Moreover, metering was conducted in an online responsive approach.

8 Experiments and Results

A set of experiments were conducted to prove the low probe effect of the deployed metering probes at the cloud front, and to show the distributed proc filesystem transport protocol's low overhead. Moreover, a comprehensive study of the effect of different metering parameters on the resources usage by the framework's different metering engines is presented.

We have adopted General Linear Models (GLM) [5, 20, 21], and Analysis of Variance (ANOVA) [15, 20] statistical models to analyze the results of the experiments. In experiments with large population sizes, like in our case, ANOVA prerequisites are very hard to establish; data normality and homogeneity, yet the Central Limit Theory (CLT) [7, 9] states that population tends to be normal as its size gets larger. Large population sizes are defined as >30 or >50 according to [8, 18], which are very small compared to our experiments sample sizes being in the range of thousands. We have coupled GLM with ANOVA to confirm and verify the results through more than one modeling perspective that lead to the same indications. The following three experiments were conducted to cover all performance aspects for the proposed metering framework.

8.1 Distributed Proc Filesystem vs. TCP

In this experiment the distributed proc filesystem protocol is compared to TCP. The experiment environment is built up of one correlation VM and 12 collection VMs. The distributed proc filesystem is compared to an Apache web server deployed on the correlation engine and metering data are submitted by the collection engines using HTTP/TCP.

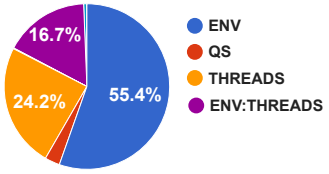


Figure 11: PROC vs. Apache - GLM/ANOVA results

Two metering factors are used in the experiment runs, namely the queue size and the number of concurrent threads represented by collection engines. The queue size ranged from 40 KB to 200 KB, and the number of concurrent threads ranged from 2 to 12. A third factor is defined as the environment; TCP or PROC. The yield of the experiment is defined as the kernel CPU usage by the Proc kernel module and the Apache web server.

An ANOVA/GLM factorial experiment was used to identify the influential factors. The pie chart in Figure 11 shows the effect of the environment on the yield which is of magnitude 55.4% of the overall effects magnitude. The 3D column chart in Figure 12 shows the kernel CPU usage in milliseconds for

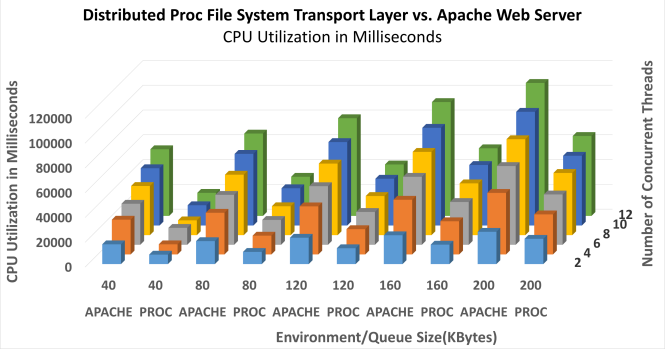


Figure 12: PROC vs. Apache - CPU utilization

each experiment run. A considerable saving, up to 35%, of CPU utilization is achieved by the distributed proc filesystem transport protocol, which reflects the amount of probe effect saving. This came on the expense of speed as illustrated by the 3D column chart in Figure 13.

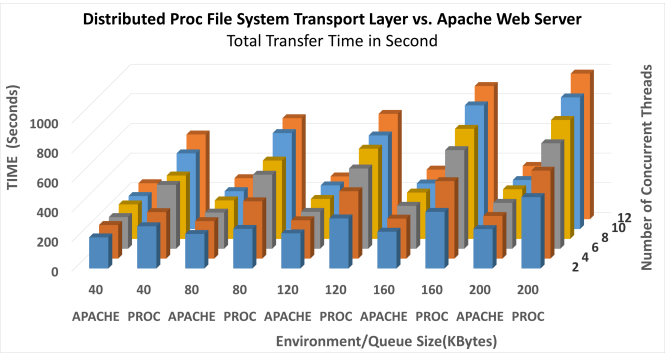


Figure 13: PROC vs. Apache - speed

8.2 Framework End-to-End GLM/ANOVA Experiment

The End-to-End GLM/ANOVA factorial experiment is designed to study the effect of different metering attributes and factors. The table in Figure 14 summarizes the different experiment factors, factor levels, and the affected metering engines.

Factor	Abbreviation	Levels	Units	Metering Engines
Queue Size	QS	40, 80, 120, 160, 200	Kbytes	All Engines
Inter-Collect Gap	ICG	2, 4, 6, 8	Seconds	All Engines
Number of Physical Nodes	PN	1, 2, 3, 4, 5	Nodes	None
Number of virtual Machines	VMs	6,12,18,24,30	VMs	None
Number Of Resources	R	2,24	Resources	Collection Engines
		48, 96, 144, 192, 240	Resources	Correlation, Storage, Billing, and PROC
Correlation Duration	CD	60, 120, 180	Seconds	Correlation, Storage, Billing, and PROC
Correlation Mode	SE	Epoch, Adhoc	Boolean/ Categorical	Correlation, Storage, Billing, and PROC

Figure 14: GLM/ANOVA experiments factors

The following is the GLM/ANOVA results of the defined factors against different yields.

8.2.1 CPU. In Figure 15, plots (A), (B), and (C) show that the inter-collect-gap and the number of resources are the most influential factors on the collection, correlation and storage engine CPU usage respectively. Plot (C) shows a small yet influential effect of the correlation duration on the CPU usage of the storage engine. Plot (D) shows that the correlation duration has the highest influence on the billing engine CPU usage followed by the number of resources. As the number of resources increases and the inter-collect-gap decreases the number of CMOs generated by the collection engines gets bigger and hence requires more CPU computing resources to process on all metering fronts. As the correlation duration decreases the number of correlation CMOs increases and hence the metering processing requires more CPU resources by the back-end services.

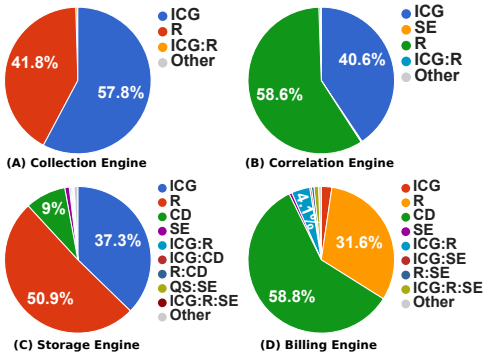


Figure 15: GLM/ANOVA CPU usage results

As per the column charts in Figure 16, Plot (A) and (B) show the inversely proportional effect of the inter-collect-gap on the collection and the correlation engines CPU usage respectively. Plot (A) shows the very low usage of the collection engines which does not exceed 1.7% at the lowest level of inter-collect-gap indicating the very low probe effect. Plot (B) shows the considerable high needs of the CPU by the correlation engine. Plot (C) and (D) shows the maximum processing time of a CMO which does not exceed 10 milliseconds; the CMO undergoes two correlation processes, one for the corresponding branch correlation plugin and the other for the metering dome. Plot (E) and (F) show the very low CPU needs by both the storage and the billing engines.

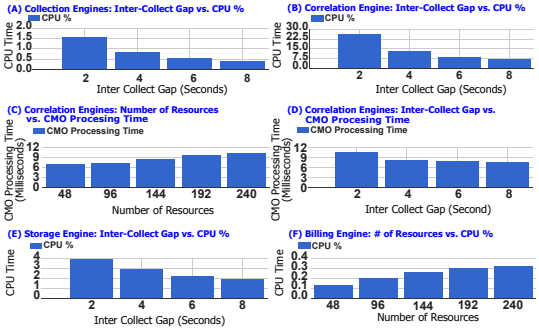


Figure 16: CPU usage column charts

The column chart in Figure 17 shows the correlation processing time vs. the correlation duration at different inter-collect-gap settings. At lower inter-collect-gap, CMOs overflow occurs where the correlation engine cannot process the CMOs at high rates of generation. The inter-collect-gap should be chosen carefully and the adaptive correlation mode should be used to saturate such symptoms. The importance of this graph is that it emphasis the design decisions behind the distributed proc filesystem, showing that the lower CPU usage favored over data transfer speed is the correct decision as the low speed of the distributed proc filesystem can generate CMO rates that are much higher than the processing rates of the correlation engines.

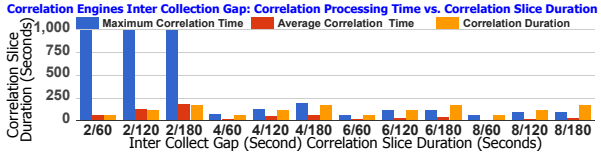


Figure 17: Correlation duration time vs. correlation duration

8.2.2 Memory. The pie charts in Figure 18 show the different factors effect on the memory usage by different metering engines. The number of resources is the most influential factor on the collection engine memory usage as per plot (A). Plot (B) shows that the correlation mode has the highest influence on the correlation engine memory usage, as in case of epoch-based correlation a lot of memory buffers are allocated to store epoch CMOs and cache them for submission to the back-end. Plots (C) and (D) show that the correlation duration is the most influential factor on the storage and the billing engines respectively; smaller correlation durations lead to larger numbers of correlation and billing CMOs.

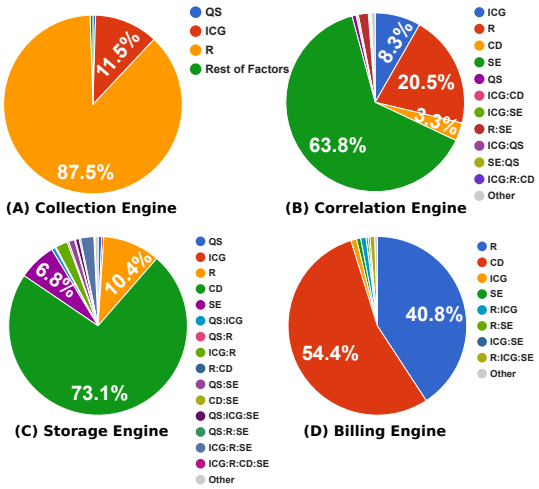


Figure 18: GLM/ANOVA memory usage results

As per the column charts in Figure 19, plot (A) shows the low memory usage by the collection engine which did not exceed 2.5%, showing a very low probe effect. Plot (B) shows the large memory needs by the correlation engines when configured to use epoch-based correlation. Plot (C) shows high memory usage by the storage engine as huge numbers of collection CMOs were configured for storage. Plot (D) shows the low memory needs of the billing engine as the number of billing transactions are a function in the number of correlation CMOs.

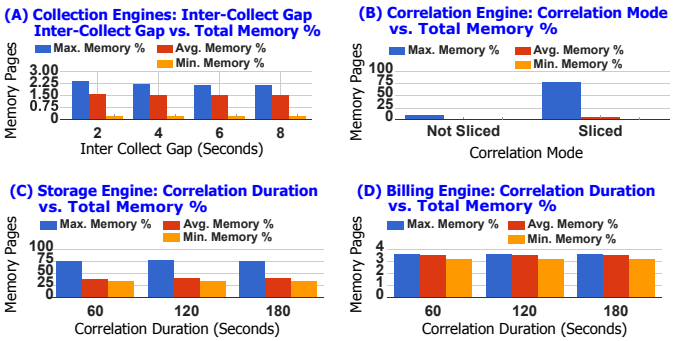


Figure 19: Memory usage column charts

8.2.3 I/O. The pie charts in Figure 20 show the different factor effects on I/O usage by different engines. All the plots show that the number of resources and the inter-collect-gap are the most influential factors by all metering engines. The correlation duration has a minor effect on the storage engine as per plot (C).

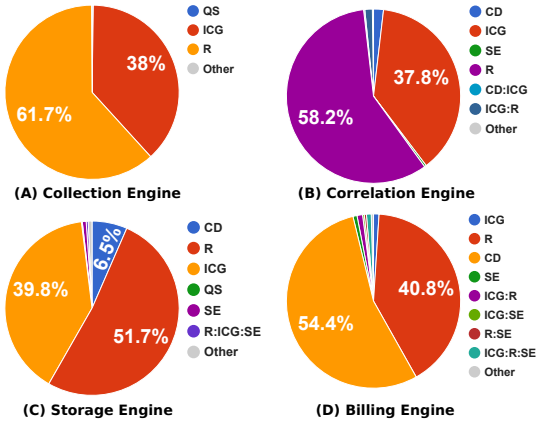


Figure 20: GLM/ANOVA I/O usage results

8.2.4 Data Transfer. The effective data transfer size represents the transferred CMOs being used effectively by the correlation engine. The overall data transfer is the transfer of all the CMOs between the collection and correlation engines including duplicates resulting from TTL misprediction. Mispredictions result from delays in the metering collection process, whose actual runtime durations do not match the heuristics of assessing the correct time for reading new data from the proc, and hence result in reading the same proc file entry CMOs twice. This is detected by the correlation engine but in any case the data needs to be read to identify its oldness.

The pie charts in Figure 21 show the GLM/ANOVA influential factors related to the data transfer. Plots (A) and (B) show that the number of resources and inter-collect-gap have the most influential effect on the data transfer size. The close values of the two pie chart shows the uniform behavior of the

prediction. The results can be explained in light of the fact that more CMOs are generated as the number of resources gets larger, and as the inter-collect-gap is set to smaller durations. Plot (C) shows that the queue size is the most influential factor on the speed of the data transfer. This can be explained in light of the way the distributed proc filesystem protocol is designed, which is based on packet group commit, and hence as the queue size gets bigger a larger number of packets is committed through fewer acknowledgement packets.

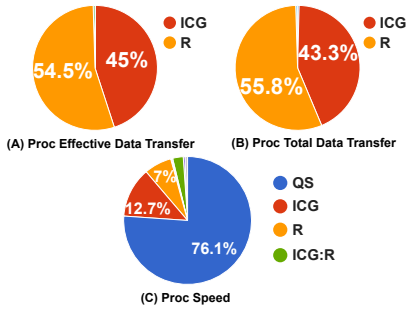


Figure 21: GLM/ANOVA data transfer results

The column charts in Figure 22 give more insight into the data transfer results. The charts (A) and (B) show the data transfer against the inter-collect-gap and the number of resources. The blue columns represent the total data transfer and the red columns represent the effective data transfer. The yellow columns are the calculated size of data transfer if TTL was not used. The plots show the considerable amount of network bandwidth saved by applying the TTL prediction mechanism. Plot (C) shows the directly proportional effect of the queue size on the data transfer speed and plot (D) shows the inversely proportional effect of the inter-collect-gaps on the data transfer speed. The plots show a maximum of 0.6 MB/sec data transfer bandwidth at the most aggressive metering configuration, which does not exceed 0.48% of the total bandwidth on a 1 Gb/sec network backbone.

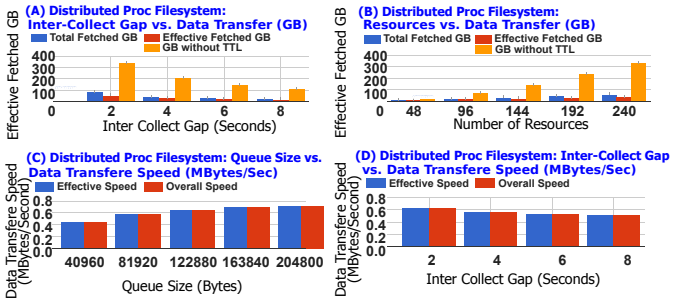


Figure 22: Data transfer column charts

The 3D surface diagram presented in Figure 23 shows the relation between the prediction accuracy and the number of proc file entries, queue sizes, and the inter-collect-gaps. In general, a directly proportional relationship exists between the three factors and the prediction accuracy. We evaluate the accuracy of the prediction as the percentage of the data that is being used out of the overall data, which in our experiments range from

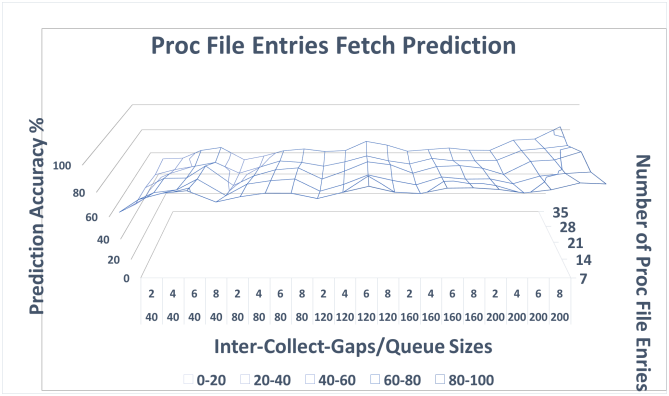


Figure 23: TTL prediction accuracy

52.79% to 87.05%. This misprediction rate is acceptable as the collectors can be delayed in some cases that make them unable to meet their deadline promises.

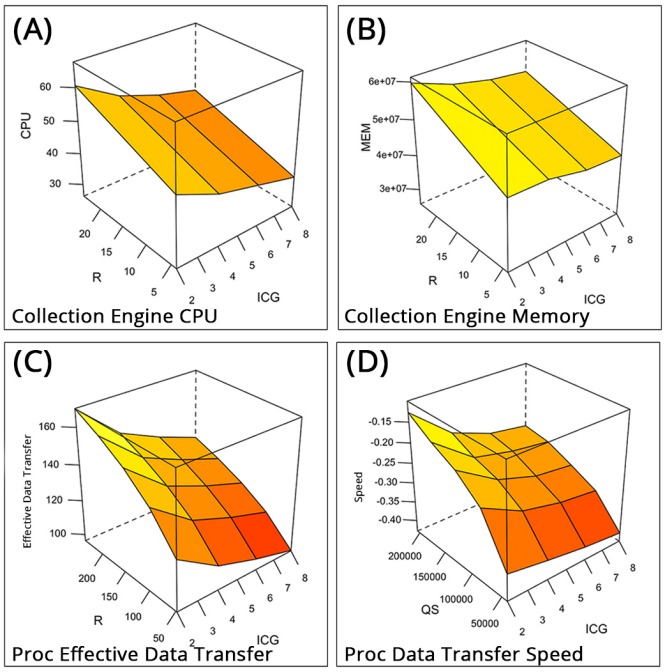


Figure 24: 3D interaction diagrams

The probe effect is most crucial at the collection engine and the distributed proc filesystem transport as it can affect the running cloud services. The 3D interaction diagrams in Figure 24 show the interaction between the most influential factors. Plots (A), (B), and (C) show the directly proportional effect of the number of resources and the inversely proportional effect of the inter-collect-gap on the collection engine CPU and Memory usage, and the effective data transfer respectively. Plot (D) shows the directly proportional effect of the queue size and the inversely proportional effect of the inter-collect-gap on the data transfer speed. All the plots are aligned with the GLM/ANOVA results presented earlier.

8.3 Online Shop Web Application Metering Plugin

The objective of this experiment is to investigate the effect of the metering plugin deployment on the application end user service experience. The metering plugin is deployed once on the local target virtual machine, and in another setup on a separate virtual machine configured to handle all the Apache web servers of a single branch. The factors investigated are the number of concurrent users, the transaction type, and the metering mode. The number of concurrent users ranged from 2 to 20 per branch. Different web transaction types where used, namely products listing, product details view, add product to shopping cart, and checkout shopping cart. And the metering mode represents the metering plugin deployment option, namely not deployed, deployed locally, and deployed remotely. The yield of the experiment is the transaction response time in seconds. The WLG is used to apply web requests, different workloads, on the shop environment.

The pie chart in Figure 25 shows that the influence of the metering plugin deployment option on the response time is 0.6% which is of minimal negligible effect. The column chart in Figure 26 shows the average transaction response time in seconds with respect to the number of concurrent users and metering plugin deployment. The chart illustrates the negligible minor difference between the three different deployment options with respect to the number of concurrent users. The presented results are aligned with the ANOVA.GLM results.

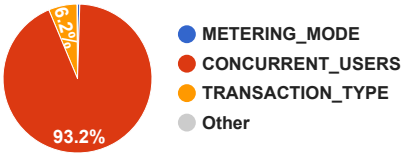


Figure 25: GLM/ANOVA shop metering plugin results

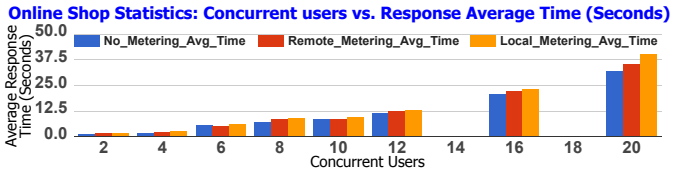


Figure 26: Shop web transaction average response time

9 Conclusion and Future Work

In this paper, a unified cloud metering framework was presented based on a data modeling approach. An extensible data representation is demonstrated through an object oriented extensible Cloud Metering Markup Language (CMML), which contributed to the highly shareable characteristics of the model. The proposed framework is programmable and extensible, enabling the metering of cloud resources at various levels of abstractions with ease through the flexibility of writing code. The key design decision adopted is to deal with metering objects rather than flat passive data. The introduction of autonomous mobile CMOs and object receptors unlocked a lot of desired features whereby the metering data are coupled with their corresponding operations. The framework is capable of presenting the underlying deployment metering architecture dynamically through the object receptors definition.

The features of the framework were demonstrated through a prototype based on a CMML interpreter implementation and a distributed proc filesystem acting as a communication backbone. The main objective of the prototype is to show one practical implementation of the proposed framework specifications. The prototype was deployed on a realistic online shopping environment to demonstrate the multi-perspective online responsive metering results generated through the distributed processing of metering data, and presented at different levels of abstraction. Moreover, the effect of workload and pricing on resources usage and billing was demonstrated through a set of functional experiments. GLM/ANOVA performance experiments were conducted to study the different factors and parameters affecting the performance of the proposed metering framework engines, and to demonstrate the considerable low probe effect induced on the cloud services resources by the framework's different engines.

Our future work will be concentrating on designing a virtual bare metal deployment mechanism that utilizes virtual resource for metering engines deployment. The new approach aims at reducing the waste of resources used in the framework deployment targeting higher return on investment, lower probe effect, and better performance.

References

[1] Jennifer M. Anderson, Lance M. Berc, Jeffrey Dean, Sanjay Ghemawat, Monika R. Henzinger, Shun-Tak A. Leung, Richard L. Sites, Mark T. Vandevoorde, Carl A. Waldspurger, and William E. Weihl, "Continuous Profiling: Where have all the Cycles Gone?", *ACM Trans. Comput. Syst.*, 15(4):357–390, November 1997.

[2] A. Anwar, A. Sailer, A. Kochut, C.O. Schulz, A. Segal, and A.R. Butt, "Cost-Aware Cloud Metering with Scalable Service Management Infrastructure". *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pp. 285–292, June 2015.

[3] Alexander Barmouta and Rajkumar Buyya, "Gridbank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing", *Proceedings of the 17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003), IEEE Computer*, Society Press, pp. 22–26, 2002.

[4] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya, "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems", *CoRR*, abs/1007.0066, 2010.

[5] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters", *Commun. ACM*, 51(1):107–113, January 2008.

[6] Jiaqing Du, Nipun Sehrawat, and Willy Zwaenepoel, "Performance Profiling of Virtual Machines", *SIGPLAN Not.*, 46(7):3–14, March 2011.

[7] Shlomo Engelberg, "The Central Limit Theorem and Low-Pass Filters", *Electronics, Circuits and Systems, 2004. ICECS 2004. Proceedings of the 2004 11th IEEE International Conference on*, IEEE, pp. 65–68, 2004.

[8] A.P. Field, "Discovering Statistics using SPSS", *ISM (London, England)*, SAGE, 2005.

[9] Michael J Glencross, "A Practical Approach to the Central Limit Theorem" , *Proceedings of the second international conference on teaching statistics*, pp. 91-95. 1988.

[10] Alex Guazzelli, Michael Zeller, Wen-Ching Lin, and Graham Williams, "PMML: An Open Standard for Sharing Models", *The R Journal*, 1(1):60–65, 2009.

[11] Riccardo Gusella and Stefano Zatti, "The Berkeley UNIX 4.3 BSD Time Synchronization Protocol", *Technical Report UCB/CSD 85/250, University of California, Berkeley*, June 1985.

[12] Riccardo Gusella, Stefano Zatti, and James M Bloom, "The Berkeley UINX Time Synchronization Protocol", *UNIX Programmers Manual, 4.3 Berkeley Software Distrib.*, vol. 2C, (1986),10 pages

[13] D. Huemer and A.M. Tjoa, "A Stepwise Approach towards an Interoperable and Flexible Logging Principle for Audit Trails", *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pp. 114 –119, April 2010.

[14] Ravi Iyer, Ramesh Illikkal, Li Zhao, Don Newell, and Jaideep Moses, "Virtual Platform Architectures for Resource Metering in Datacenters", *SIGMETRICS Perform. Eval. Rev.*, 37(2):89–90, October 2009.

[15] Raj Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley, 1991.

[16] William M. Jones, John T. Daly, and Nathan DeBardeleben, "Impact of Sub-Optimal Checkpoint Intervals on Application Efficiency in Computational Clusters", *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, ACM, New York, NY, USA, pp. 276–279, 2010.

[17] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya, "Virtual Machine Power Metering and Provisioning", *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, ACM, New York, NY, USA, pp. 39–50, 2010.

[18] A. Leon-Guerrero and C. Frankfort-Nachmias, "Essentials of Social Statistics for a Diverse Society", *SAGE Publications*, 2011.

[19] linux kvm.org, "Kvm, Kernel Based Virtual Machine", `http://www.linux-kvm.org/page/Main_Page`, 2016. [Online; accessed 25-January-2016].

[20] Douglas C. Montgomery, *Design and Analysis of Experiments*, John Wiley & Sons, 2006.

[21] Raymond H Myers, Douglas C Montgomery, G Geoffrey Vining, and Timothy J Robinson, "The Generalized Linear Model", *Generalized Linear Models: With Applications in Engineering and the Sciences, Second Edition*, pp. 202–271, 2010.

[22] V. K. Naik, K. Beaty, and A. Kundu, "Service Usage Metering in Hybrid Cloud Environments", *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pp. 253–260, March 2014.

[23] F. A. Pereira da Silva, P. A. Da Mota Silveira Neto, V. Cardoso Garcia, F. A. Mota Trinta, and R. Elia Assad, "Monext: An Accounting Framework for Infrastructure Clouds", *Parallel and Distributed Computing (ISPDC), 2013 IEEE 12th International Symposium on*, pp. 26–33, June 2013.

[24] F. A. Pereira da Silva, P. A. Da Mota Silveira Neto, V. Cardoso Garcia, F .A. Mota Trinta, and R. Elia Assad. "Veloz: A Charging Policy Specification Language for Infrastructure Clouds", *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pp. 1–7, July 2013.

[25] Rosario M. Piro, Michele Pace, Antonia Ghiselli, Andrea Guarise, Eleonora Luppi, Giuseppe Patania, Luca Tomassetti, and Albert Werbrouck, "Tracing Resource Usage over Heterogeneous Grid Platforms: A prototype RUS Interface for DGAS", *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '07, IEEE Computer Society, Washington, DC, USA, pp. 93–101, 2007.

[26] QEMU.org, "Qemu, Open Source Processor Emulator", `http://wiki.qemu.org/Main_Page`, 2016. [Online; accessed 25-January-2016].

[27] Gang Ren, E. Tune, T. Moseley, Yixin Shi, S. Rus, and R. Hundt, "Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers", *Micro, IEEE*, 30(4):65 –79, July-August, 2010.

[28] Benjamin Serebrin and Daniel Hecht, "Virtualizing Performance Counters", *ACM Trans. Comput. Syst.*, 2011.

[29] F.A. Silva, P. Neto, V. Garcia, F. Trinta, and R. Assad, "Accounting Federated Clouds Based on the JITCloud Platform", *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 186–187, May 2013.

[30] T. Singh and P.K. Vara, "Smart Metering the Clouds", *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops on*, pp. 66 –71, 29 2009-july 1 2009.

[31] virtualbox.org, "Virtualbox", `https://www.virtualbox.org/`, 2016. [Online; accessed 25-January-2016].

[32] VMware, "VMware Workstation Player", `https://www.vmware.com/products/player`, 2016. [Online; accessed 25-January-2016].

[33] Miao Wang, V. Holub, T. Parsons, J. Murphy, and P. O'Sullivan, "Scalable Run-Time Correlation Engine for Monitoring in a Cloud Computing Environment", *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, pp. 29 –38, March 2010.

Karim Sobh has a Ph.D. in Computer Science from the American University in Cairo. He received his B.Sc. and M.Sc. degrees in Computer Science from the same university. Dr. Sobhs specialization is in distributed systems and cloud computing, and his PhD. topic is cloud environments metering. As a systems architecture consultant at IBM Egypt his role was to provide system architecture consultation for large projects. Currently he is the founder and an active partner in Code-Corner, a software development firm providing software development, subcontracted services, cloud deployment services, consultation services, and turn-key solutions using open source technologies.

Amr El-Kadi is Professor and former Chair, the Computer Science and Engineering Department at the American University in Cairo. He was a member of the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP) that developed the Software Engineering Code of Ethics and Professional Practices. Before joining AUC he was a consulting engineer with the Information, Technology and Facilities Department at the World Bank, Washington DC. He received his D.Sc. degree in Electrical Engineering and Computer Science from The George Washington University. Dr. El-Kadi is a Senior Member of IEEE (serving as the Middle East Representative of the IEEE Technical Committee on Operating Systems and Applications Environments), a member of ACM, and a member of Eta Kappa Nu (the US National Electrical and Computer Engineering Honor Society).

Instructions for Authors

The International Journal of Computers and Their Applications is published multiple times a year with the purpose of providing a forum for state-of-the-art developments and research in the theory and design of computers, as well as current innovative activities in the applications of computers. In contrast to other journals, this journal focuses on emerging computer technologies with emphasis on the applicability to real world problems. Current areas of particular interest include, but are not limited to: architecture, networks, intelligent systems, parallel and distributed computing, software and information engineering, and computer applications (e.g., engineering, medicine, business, education, etc.). All papers are subject to peer review before selection.

A. Procedure for Submission of a Technical Paper for Consideration

1. Email your manuscript to the Editor-in-Chief, Dr. Fred Harris, Jr., Fred.Harris@cse.unr.edu.

2. Illustrations should be high quality (originals unnecessary).

3. Enclose a separate page (or include in the email message) the preferred author and address for correspondence. Also, please include email, telephone, and fax information should further contact be needed.

B. Manuscript Style:

1. The text should be **double-spaced** (12 point or larger), **single column** and **single-sided** on 8.5 X 11 inch pages.
2. An informative abstract of 100-250 words should be provided.
3. At least 5 keywords following the abstract describing the paper topics.
4. References (alphabetized by first author) should appear at the end of the paper, as follows: author(s), first initials followed by last name, title in quotation marks, periodical, volume, inclusive page numbers, month and year.
5. Figures should be captioned and referenced.

C. Submission of Accepted Manuscripts

1. The final complete paper (with abstract, figures, tables, and keywords) satisfying Section B above in **MS Word format** should be submitted to the Editor-in-Chief.
2. The submission may be on a CD/DVD or as an email attachment(s) . **The following electronic files should be included:**
    - Paper text (required).
    - Bios (required for each author). Integrate at the end of the paper.
    - Author Photos (jpeg files are required by the printer, these also can be integrated into your paper).
    - Figures, Tables, Illustrations. These may be integrated into the paper text file or provided separately (jpeg, MS Word, PowerPoint, eps).

3. Specify on the CD/DVD label or in the email the word processor and version used, along with the title of the paper.

4. Authors are asked to sign an ISCA copyright form (http://www.isca-hq.org/j-copyright.htm), indicating that they are transferring the copyright to ISCA or declaring the work to be government-sponsored work in the public domain. Also, letters of permission for inclusion of non-original materials are required.

Publication Charges

After a manuscript has been accepted for publication, the contact author will be invoiced for publication charges of **$50.00 USD** per page (in the final IJCA two-column format) to cover part of the cost of publication. For ISCA members, $100 of publication charges will be waived if requested.

January 2014