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

INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

Page

Real-Time Systems Scheduling of Multiple Virtual Machine 91
Christine Niyizamwiyitira and Lars Lundberg

An Enhancement in Segmentation of Magnetic Resonance Images of Brain Tumors Using Symmetry and Active Contour 110
Mubbashar Saddique, Kalim Qureshi, Jawad Haider Kazmi, and Zainab Meraj

Using an Uncalibrated Camera for Undistorted Projection over a Mobile Region of Interest 120
Mamona Awan and Kwang Hee Ko

CMML: A Cloud Metering Markup Language 133
Karim Sobh and Amr El-Kadi

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

Volume 24, No. 3, Sept. 2017 **ISSN 1076-5204**

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

EDITOR-IN-CHIEF

Dr. Frederick C. Harris, Jr., Professor
Department of Computer Science and Engineering
University of Nevada, Reno, NV 89557, USA
Phone: 775-784-6571, Fax: 775-784-1877
Email: Fred.Harris@cse.unr.edu, Web: http://www.cse.unr.edu/~fredh

ASSOCIATE EDITORS

Dr. Hisham Al-Mubaid
University of Houston-Clear Lake, USA
hisham@uhcl.edu

Dr. Antoine Bossard
Advanced Institute of Industrial Technology, Tokyo, Japan
abossard@aiit.ac.jp

Dr. Mark Burgin
University of California,
Los Angeles, USA
mburgin@math.ucla.edu

Dr. Sergiu Dascalu
University of Nevada, USA
dascalus@cse.unr.edu

Dr. Sami Fadali
University of Nevada, USA
fadali@ieee.org

Dr. Vic Grout
Glyndŵr University,
Wrexham, UK
v.grout@glyndwr.ac.uk

Dr. Yi Maggie Guo
University of Michigan,
Dearborn, USA
magyiguo@umich.edu

Dr. Wen-Chi Hou
Southern Illinois University, USA
hou@cs.siu.edu

Dr. Ramesh K. Karne
Towson University, USA
rkarne@towson.edu

Dr. Bruce M. McMillin
Missouri University of Science and Technology, USA
ff@mst.edu

Dr. Muhanna Muhanna
Princess Sumaya University for Technology, Amman, Jordan
m.muhanna@psut.edu.jo

Dr. Mehdi O. Owrang
The American University, USA
owrang@american.edu

Dr. Xing Qiu
University of Rochester, USA
xqiu@bst.rochester.edu

Dr. Abdelmounaam Rezgui
New Mexico Tech, USA
rezgui@cs.nmt.edu

Dr. James E. Smith
West Virginia University, USA
James.Smith@mail.wvu.edu

Dr. Shamik Sural
Indian Institute of Technology
Kharagpur, India
shamik@cse.iitkgp.ernet.in

Dr. Ramalingam Sridhar
The State University of New York at Buffalo, USA
rsridhar@buffalo.edu

Dr. Junping Sun
Nova Southeastern University, USA
jps@nsu.nova.edu

Dr. Jianwu Wang
University of California
San Diego, USA
jianwu@sdsc.edu

Dr. Yiu-Kwong Wong
Hong Kong Polytechnic University,
Hong Kong
eeykwong@polyu.edu.hk

Dr. Rong Zhao
The State University of New York
at Stony Brook, USA
rong.zhao@stonybrook.edu

ISCA Headquarters···•···P. O. Box 1124, Winona, MN 55987 USA···•···Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: http://www.isca-hq.org

Copyright © 2017 by the International Society for Computers and Their Applications (ISCA)
All rights reserved. Reproduction in any form without the written consent of ISCA is prohibited.

Real-Time Systems Scheduling of Multiple Virtual Machines

Christine Niyizamwiyitira* and Lars Lundberg*
Blekinge Institute of Technology, SE-379-71 Karlskrona, SWEDEN

Abstract

The use of virtualized systems is growing, and one would like to benefit from this kind of systems also for real-time applications with hard deadlines. There are two levels of scheduling in real-time applications executing in a virtualized environment: traditional real-time scheduling of the tasks in the real-time application inside a Virtual Machine (VM), and scheduling of different VMs on the hypervisor level. Traditional real-time scheduling uses methods based on periods, deadlines and worst-case execution times of the real-time tasks. In order to apply the existing theory also to virtualized environments we must obtain periods and (worst-case) execution times for VMs containing real-time applications. In this paper, we describe a technique for calculating periods and execution times and utilization for VMs containing real-time applications with hard deadlines. We show that when we look at all VMs that share a physical processor we are able to use longer (better) periods. Alternatively, if the periods are the same, we are able to use a smaller amount of the processor resource for the VMs and more tasks become schedulable compared to when we look at each VM in isolation. We also introduce an overhead model that makes it possible to find VM periods that minimize the processor utilization.

Key Words: Real-time virtual machine; real-time scheduling; hard deadlines; VM overhead; VM period.

1 Introduction

There is a strong trend towards virtualization of computer systems, and one would like to also run real-time systems in virtualized environments. However, moving a real-time system with hard deadlines to a virtualized environment where a number of Virtual Machines (VMs) share the same physical computer is a challenging task. The original real-time application was designed such that all tasks were guaranteed to meet their deadlines provided that the physical computer was fast enough. In a system with faster processors, and more cores, one would like to put several VMs on the same physical hardware and some or all of these VMs may contain real-time tasks with hard deadlines. In order to take full advantage of the hardware, more than one VM may share a processor core. This is the scenario that we consider in this study, i.e. k VMs share the same processor core, and each VM contains a real-time application. We assume that for each core, the identities of the VMs that share that core are known. We also assume that these VMs are scheduled to the physical processor core using static priorities. In such a system there will be scheduling on two levels [1, 24]. The first level is traditional real-time scheduling of the tasks within a VM. The second level is scheduling of VMs by the hypervisor; the hypervisor controls several VMs on the same physical hardware.

Two classic real-time scheduling algorithms are Rate Monotonic Scheduling (RMS) where tasks are assigned static priorities based on deadlines, and Earliest Deadline First (EDF) where task priorities are dynamic. These kind of scheduling algorithms enable to guarantee certain real-time properties in non-virtualized systems. These scheduling algorithms are based on the periodic behavior of the real-time tasks, i.e. each task has a period T and a worst-case execution time C. This means that a task may in the worst-case need to use the processor for C time units during each period, the length of the period is T time units. In order to use existing real-time scheduling theory also on the hypervisor level, i.e. when scheduling different VMs on the physical hardware, we need to calculate a period T_{VM} and a worst-case execution time C_{VM} for each VM such that all real-time tasks in the VMs will meet their deadlines.

Previous work [20] has found a method for calculating an execution time C_{VM} and a period T_{VM} for a VM such that all real-time tasks in the VM will meet their deadlines. That study considered each VM in isolation, i.e. without knowledge about the other VMs sharing the processor. The contribution in this paper is that we define an improved execution time C_{VM_i} and period T_{VM_i}, by considering a holistic perspective, i.e., we consider the whole work-load of all VMs that share a processor core.

The holistic approach gives more information about the work-load and does not require to be overly pessimistic, and as a result more real-time programs become schedulable. We also define C_{VM_i} and T_{VM_i} in the presence of overhead for context switches between VMs.

2 Background and Related Work

Real-time scheduling theory (or non-virtualized systems shows that the minimum processor utilization for which a

---

*Department of Computer Science and Engineering, Faculty of Computing. Email: Christine.niyizamwiyitira@bth.se, lars.lundberg@bth.se.

ISCA Copyright© 2017

periodic real-time system can miss a deadline, using fixed priority scheduling, i.e. using RMS, decreases as the number of processors increases, e.g. 69.3% for one processor systems [16], and 53.2% for two processor systems and then down to as little as 37.5% for systems with infinitely many processors [18]. Consequently, compared to multiprocessor systems, the processor utilization is generally higher for systems with one processor. This is one reason why we have assumed that each core of a multi-core processor contains a number of VMs and each VM that contains a real-time application has only one virtual processor. Also, most existing real-time applications are developed for systems with one processor.

An additional advantage of just having one virtual core in each VM is that one can bind each VM to a physical core, thus minimizing unpredictable dynamic cache effects, i.e., the processor cache will be cold (empty) when a VM is migrated from one core to another. Such effects become problematic in real-time systems since applications with hard deadlines need to control the worst-case behavior. We, therefore, expect that one future way of using virtualization will be that a VM containing a real-time application will be bound to a processor core a modern multi-core hardware server. In order to provide high hardware utilization, we expect that many VMs may share the same processor core.

Very few studies have explicitly focused on hard real-time scheduling in virtualized systems. Some results on real-time tasks with soft deadlines have been studied with the focus on real-time hypervisor scheduling framework for Xen [12, 29]. There are a number of results concerning so called proportional-share schedulers [9, 22, 27]. These results looked at a real-time application that runs inside an operating system process. The proportional-share schedulers divide the processor resource in predefined proportions to different processes. However, none of these results explicitly address hard real-time issues such as worst-case scenarios, periods/deadlines and worst-case execution times. In [17], the authors looked at a model for deciding which real-time tasks to discard when the cloud system's resources cannot satisfy the needs of all tasks. That model does, however, not address the problems associated with hard deadlines. The VSched system, which runs on top of Linux, provides soft real-time scheduling of VMs on physical servers [14]. However, the problems with hard deadlines are not addressed in that system.

In the area of hierarchical scheduling, there have been plenty of studies. In [10], authors proposed a hierarchical real-time virtual resource model that permits resource partitioning to be extended to multiple levels (similar to the two-level scheduling situation in virtualized systems). In [25, 26] the authors proposed a resource model for hierarchical schedulers to characterize a periodic resource allocation and present exact schedulability conditions under RMS and EDF algorithms. This method derives timing requirements of a parent scheduler from the timing requirements of its child scheduler in a compositional manner such that the timing requirement of the parent scheduler is satisfied if and only if the child scheduler is satisfied. Later on, they proposed a compositional real-time scheduling framework with a periodic model that enables a group of real-time applications to be a single real-time resource requirement to the upper level scheduler. These scheduling schemes help to schedule large complex systems by breaking them down into subsystems.

In [8, 15] the authors studied a two-level hierarchical architecture to schedule many applications on a single processor. Each application is associated with a server and each server is assigned a portion of the processor [15]. There is a global scheduler that determines which application, i.e., which server, should be allocated to the processor at any given time and a local scheduler that determines which of the chosen application's tasks should actually execute. Both schedulers use fixed priority pre-emptive scheduling policy. All of the hierarchical scheduling results mentioned above consider each VM in isolation, i.e., no study takes a holistic approach where the entire set of VMs are considered. Previous results on age-constraint real-time tasks (in a uniprocessor environment) show that one can guarantee the schedulability for more cases when the entire work-load is taken into consideration [19].

In [6] the authors studied a reservation-based algorithm, i.e., a constant bandwidth server (CBS) on top of EDF for scheduling real-time tasks with hard deadlines on VMs. A reservation-based scheduler allocates a computation budget for every reservation period to each VM. The execution of a VM does not depend on the other VMs running on the same hardware (temporal isolation), rather it depends only on task's period and execution time. The results show that VM technology and scheduling algorithm can affect the real-time application performance. They propose to use less pessimistic analysis to dimension the VM scheduling parameters if one uses CBS algorithm. Interaction between VMs is not considered whereas in this paper, we consider also the interaction between VMs therefore VMs priority is set accordingly.

In [13] the authors developed a Compositional Scheduling Architecture (CSA) that is built on the Xen virtualization platform. The architecture allows timing isolation among virtual machines and supports timing guarantees for real-time tasks running on each virtual machine. The study uses a pessimistic approach where every VMs is treated in isolation, whereas in this paper every VM is treated with respect to other VMs that they share resources. In [21] the authors present a model that include the cache related in hierarchical scheduling while keeping temporal isolation between applications that share a single processor, yet interaction between VMs is not considered.

In [7] the authors propose a mechanism to schedule soft real-time systems, which provides a temporal isolation between VMs that share a CPU. In this paper we consider even when a VM is affected by the work-load from other VMs that they share resources, a hard-real-time system with a strict deadline is considered.

In [5, 23] the authors present a model that accounts for the overhead, in the compositional hierarchical scheduling for uniprocessor however, the entire work-load was not considered. In [30] compositional scheduling theory was also applied for multi-core VM scheduler for Xen real-time virtualization platform. However, task's migration across processor/core in the same VM which causes significant overhead was neglected.

In connection with hard deadlines systems, the utilization has been studied for real-time systems that respond to an external environment within a specific deadline that is called age constraint. This age constraint is the time between the beginning of the execution of a task in one period and the end of the task in the next period [19]. In this paper we use the idea from the age constraint approach for scheduling real-time tasks in the VMs.

In [20] the authors addressed the problem of scheduling hard real-time applications in a VM. The authors proposed a technique such that real-time applications could meet their deadlines when they are scheduled on a single VM. In this paper we improve this technique by proposing a method that schedules many VMs as whole instead of looking at each VM in isolation. We should not ignore the overhead brought by VMs, a case that considers this is also presented herein. The method described in this paper saves the resource utilization while scheduling many VMs.

3 Problem Definition

We consider the case when k VMs share the same processor core (see Figure 1(i)); all VMs have one virtual processor. We assume that for each core the identities of the VMs that share a processor core is known. We also assume that these VMs are scheduled to the physical core using static priorities. Each VM_i $(1 \le i \le k)$ runs a real-time program that consists of n_i tasks $\tau_{i,j}$ $(1 \le j \le n_i)$, i.e. $\tau_{i,j}$ denotes task j in VM_i. A task $\tau_{i,j}$ is defined by its worst-case execution time $C_{i,j}$ and period $T_{i,j}$ [4]. Since we assume that the priority follows rate monotonic scheduling (RMS), the tasks are ordered such that $T_{i,j} \le T_{i,j+1}$. This means that inside VM_i, task $\tau_{i,1}$ has the highest priority, i.e., it is never interrupted by any other task.

We assume that each task is independent and does not interact with other tasks. We also assume that the first invocation of a task is unrelated to the first invocation of any other task, i.e., we make no assumptions regarding the phasing of tasks with equal or harmonic periods. Since we assume that the deadline $D_{i,j}$ is equal to the period $T_{i,j}$, we only need two parameters for each task: $T_{i,j}$ and $C_{i,j}$ [4].

For each VM that share a physical core, we need to calculate a period T_{VM_i} and an execution time C_{VM_i} such that all tasks $\tau_{i,j}$ will meet their deadlines when VM_i executes at least C_{VM_i} time units every T_{VM_i} period.

In [20] the authors found a method for calculating an execution time C_{VM} and a period T_{VM} when one looks at a VM in isolation.

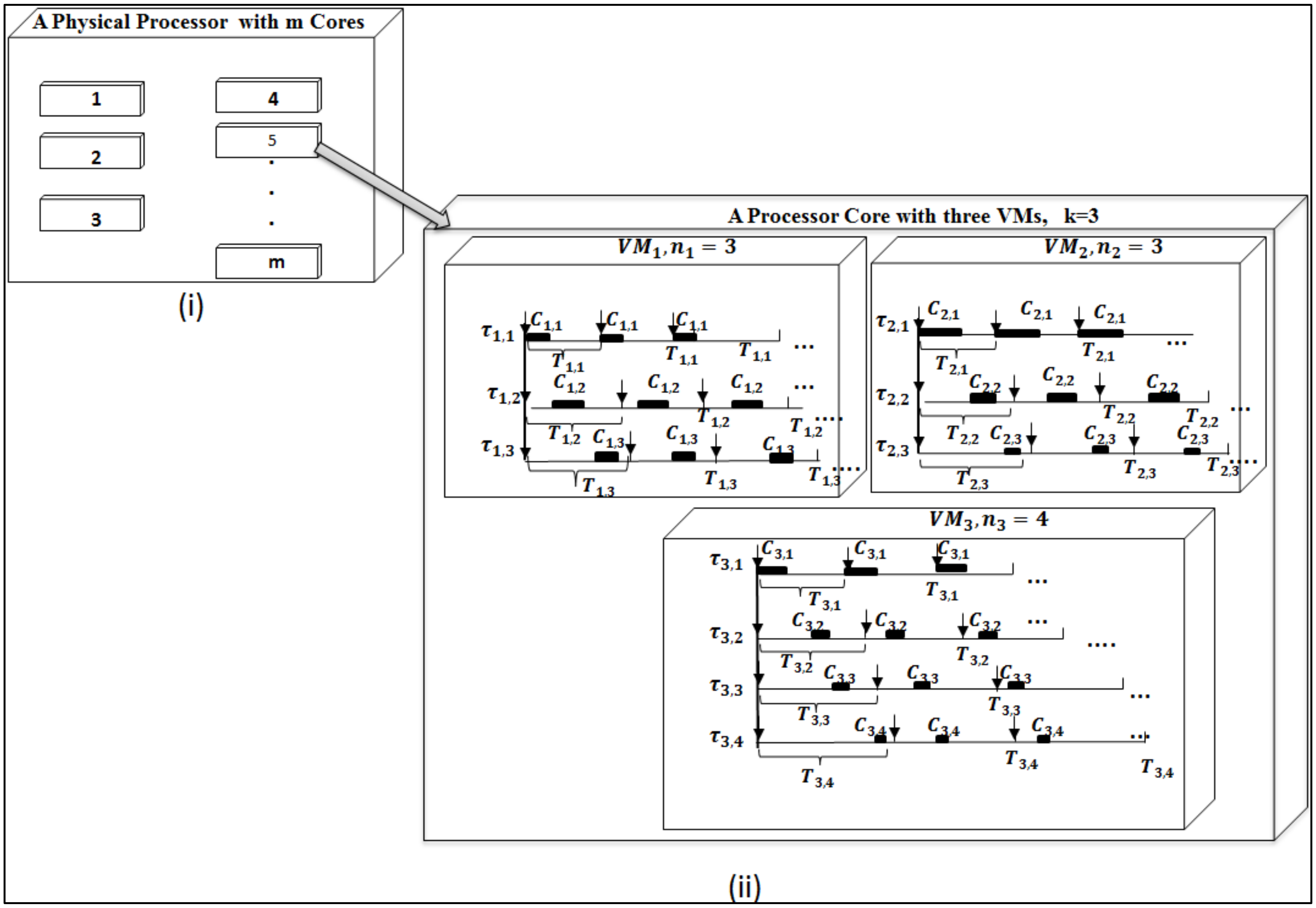

A Physical Processor with m Cores
1
2
3
4
5
m
(i)
A Processor Core with three VMs, k=3
$VM_1, n_1 = 3$
$VM_2, n_2 = 3$
$VM_3, n_3 = 4$
$\tau_{1,1}$ $\tau_{1,2}$ $\tau_{1,3}$
$\tau_{2,1}$ $\tau_{2,2}$ $\tau_{2,3}$
$\tau_{3,1}$ $\tau_{3,2}$ $\tau_{3,3}$ $\tau_{3,4}$
(ii)

Figure 1(i): A physical processor with *m* cores, and (ii) three virtual machines on a processor

A main contribution in this paper is to use information about the entire set of VMs sharing a core to reduce the resource utilization C_{VM}/T_{VM} compared to considering each VM in isolation (we will also extend the previous result by introducing an overhead model).

We also assume that we use static priorities on the hypervisor level. VM_1 has the highest priority and cannot be interrupted by any other VM, and VM_2 has the second highest priority. Figure 1(ii) shows a processor core that runs VM_1, VM_2 and VM_3. VM_1 has three tasks $\tau_{1,1}$, $\tau_{1,2}$, $\tau_{1,3}$. VM_2 has also three tasks $\tau_{2,1}$, $\tau_{2,2}$, $\tau_{2,3}$, and VM_3 has four tasks $\tau_{3,1}$, $\tau_{3,2}$, $\tau_{3,3}$, $\tau_{3,4}$. A real-time task may miss its deadline if the VM containing the task is not scheduled for execution by the hypervisor during a certain period of time. We would like to assign long periods (i.e. T_{VM_1}, T_{VM_2}, T_{VM_3}) to each VM, since this will minimize the overhead for switching VMs. However, if the VM periods are too long, the real-time tasks in the VM may miss their deadlines. Previous results show that there is a trade-off between the length of T_{VM_i} and the utilization C_{VM_i}/T_{VM_i} that a VM needs to guarantee that all tasks meet their deadlines [20]. We would like to find combinations of periods T_{VM_i} and execution times C_{VM_i} that strike a good compromise between a limited number of context switches between VMs (i.e. long T_{VM_i}) and a low utilization C_{VM_i}/T_{VM_i} for all VMs.

In a traditional real-time application, a task $\tau_{i,j}$ will voluntarily release the processor when it has finished its execution in a cycle, and $C_{i,j}$ denotes the maximum time it may execute before it releases the processor. In our case the hypervisor will make sure that VM_i releases the processor after executing for C_{VM_i} time units in a period T_{VM_i}.

4 Defining T_{VM_i} and C_{VM_i}

Let $R_{i,j}$ denotes the maximum response time for task $\tau_{i,j}$. Using traditional RMS scheduling, the worst- case response time $R_{i,j}$ for task $\tau_{i,j}$ is given by Equation (1).

$$R_{i,j} = C_{i,j} + \sum_{m=1}^{j-1} \left\lceil \frac{R_{i,j}}{T_{i,m}} \right\rceil C_{i,m} \tag{1}$$

In order to obtain $R_{i,j}$ from Equation (1), we need to use iterative numeric methods [4]. In Figure 2, we look at one VM in isolation [20]. The main result in [20] is a method of finding T_{VM} and C_{VM} such that all tasks will meet their deadlines. The most important part of the method is a function $f^{-1}(t, T_{VM}, C_{VM})$ that maps virtual time to real (wall clock) time. Before we present the main results of this paper, we will give a short overview of the previous results that considered each VM in isolation.

Consider a time period of length t. Equation (2) denotes the number of complete periods of length T_{VM} that are covered by t for the worst-case scenario; each complete period T_{VM} has execution C_{VM} (see Figure 2).

$$\left\lfloor \frac{t - 2(T_{VM} - C_{VM})}{T_{VM}} \right\rfloor \tag{2}$$

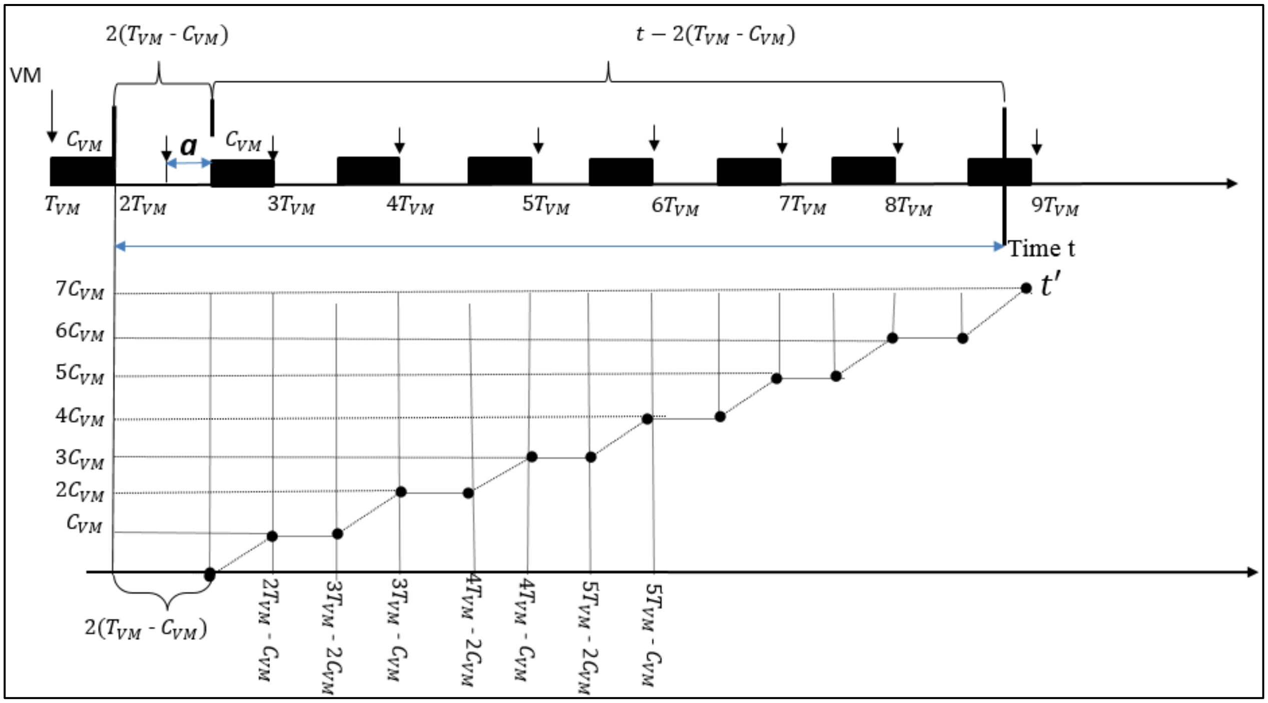


Figure 2: The worst-case scenario for an isolated VM

Let t' denotes the minimum amount of time that the VM is running during time period t. Previous results show that t' is obtained in the following way:

$$t' = max\left(0, \left\lfloor \frac{t-2(T_{VM}-C_{VM})}{T_{VM}} \right\rfloor C_{VM} + min\left(t - 2(T_{VM} - C_{VM}) - \left\lfloor \frac{t-2(T_{VM}-C_{VM})}{T_{VM}} \right\rfloor T_{VM}, C_{VM}\right)\right) \qquad (3)$$

This means that t' is a function of three parameters, i.e. $t' = f(t, T_{VM}, C_{VM})$.

For fixed T_{VM} and C_{VM}, $t' = f(t, T_{VM}, C_{VM})$ is a continuously increasing function in t, consisting of straight line segments from $\big((2+n)T_{VM} - 2C_{VM}), nC_{VM}\big)$ to$\big((2+n)T_{VM} - C_{VM}), (n+1)C_{VM}\big)$ for any n = 0, 1, 2…, (see Figure 2). As a result,

$$f(t, T_{VM}, C_{VM}) = \left\lfloor \frac{t-2(T_{VM}-C_{VM})}{T_{VM}} \right\rfloor C_{VM} + min\left(\left(t - 2(T_{VM} - C_{VM}) - \left\lfloor \frac{t-2(T_{VM}-C_{VM})}{T_{VM}} \right\rfloor T_{VM}\right), C_{VM}\right) \qquad (4)$$

The horizontal line that connects two consecutive segments represents the end of a previous execution in a period and the beginning of the next execution in the next period.

We now consider the inverse function, $f^{-1}(t, T_{VM}, C_{VM})$, i.e., $t = f^{-1}(f(t, T_{VM}, C_{VM}), T_{VM}, C_{VM})$; by looking at Figure 3, we see that $f^{-1}(t, T_{VM}, C_{VM})$ is undefined during the time intervals when $f(t, T_{VM}, C_{VM})$ is flat. In [2] the authors define the inverse of a function with flat intervals; they call it a pseudo-inverse since a function with flat intervals is not invertible. In order to handle this problem in our case we take a safe (worst-case) approach, and for all values of t in a flat interval we define $f^{-1}(t, T_{VM}, C_{VM})$ as the inverse of the maximum t value in that interval, thus making the inverse defined for all parameters t, T_{VM}, and C_{VM}.

Using the definition above, the (pseudo-)inverse of $f(t, T_{VM}, C_{VM})$ is

$$f^{-1}(t, T_{VM}, C_{VM}) = 2(T_{VM} - C_{VM}) + t + \left(\left\lceil \frac{t}{C_{VM}} \right\rceil - 1\right)(T_{VM} - C_{VM}) \qquad (5)$$

From the response time R_j (see Equation (1)) and the definition of the f^{-1}, we get the worst-case response time r_j for task τ_j :

$$r_j = f^{-1}(R_j, T_{VM}, C_{VM}) \qquad (6)$$

To meet all deadlines for all tasks τ_j, we need to select T_{VM} and C_{VM} such that

$$r_j = f^{-1}(R_j, T_{VM}, C_{VM}) \leq T_j \quad (1 \leq j \leq n_i) \qquad (7)$$

In this paper, instead of considering each VM in isolation, we take a holistic view since we know the whole work-load on the physical core. The only way that the execution can happen in the end of periods (see Figure 2) is when the core is busy, i.e., when at least one of the other VMs has a higher priority. The part indicated by "*a*" in Figure 2 is the difference between the pessimistic (considering each VM in isolation) and the optimistic (holistic) cases.

In the pessimistic case we have the same function $f^{-1}(t, T_{VM_i}, C_{VM_i})$ for all VMs; each VM can of course have

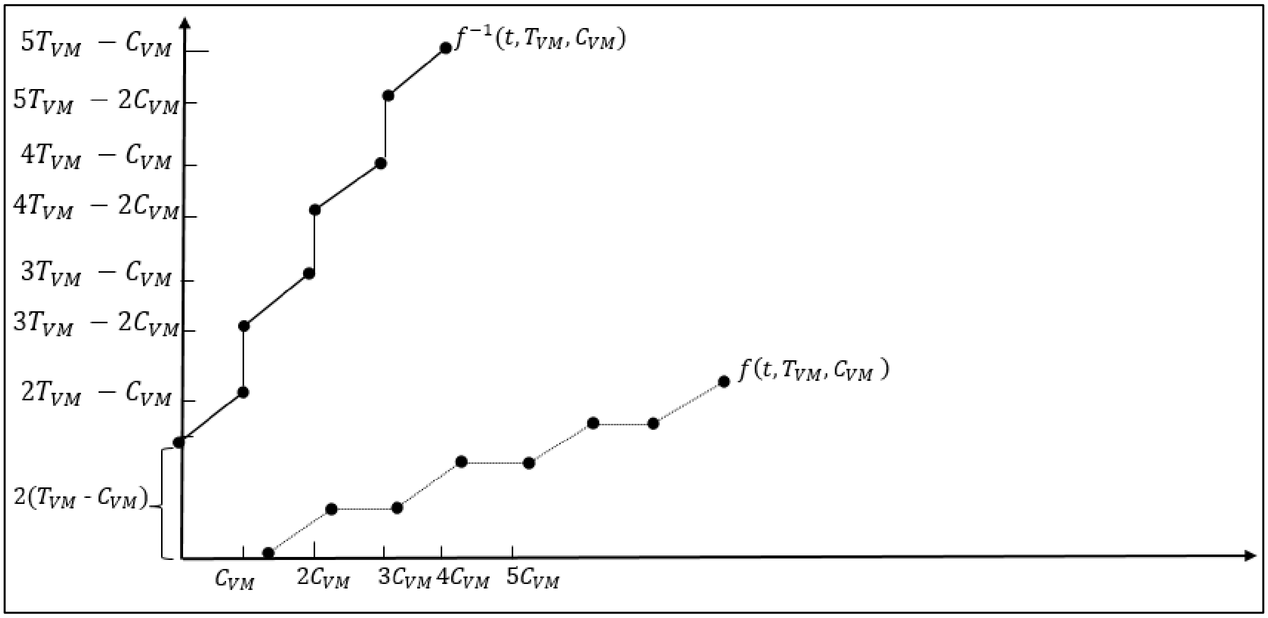


Figure 3: The (pseudo-)inverse function for an isolated VM (i.e., the pessimistic case)

their own period (T_{VM_i}) and execution time (C_{VM_i}). In the optimistic (holistic) case we have a function $f^{-1}(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$, with R_{VM_i} representing the worst-case response time for VM_i.

VMs are ordered such that VM_i has a higher priority than VM_j if $i < j$. In the worst-case scenario, all VMs are released at the same time; first of all, we consider VM_1 and take a time period of length t which may extend over several periods (see Figure 4).

In the worst-case scenario period t starts right after a period of C_{VM_1} execution that occurred at the start of a period T_{VM_1}. Since VM_1 has the highest priority, the maximum time that VM_1 has to wait for, is $T_{VM_1} - C_{VM_1}$ before it executes its first C_{VM_1}. The number of whole periods of length T_{VM_1} that is covered by t for the worst-case scenario is given in Equation (8).

$$\left\lfloor \frac{t-(T_{VM_1}-C_{VM_1})}{T_{VM_1}} \right\rfloor \tag{8}$$

Let t' denote the minimum amount of time that VM_1 is running during a time period of length t.

$$t' = max\left(0, \left\lfloor \frac{t-(T_{VM_1}-C_{VM_1})}{T_{VM_1}} \right\rfloor C_{VM_1} + min\left(\left(t - \left(T_{VM_1} - C_{VM_1}\right) - \left\lfloor \frac{t-(T_{VM_1}-C_{VM_1})}{T_{VM_1}} \right\rfloor T_{VM_1}\right), C_{VM_1}\right)\right) \tag{9}$$

For fixed T_{VM_1} and C_{VM_1}, Figure 4 shows that $t' = f(t, T_{VM_1}, C_{VM_1}, C_{VM_1})$ consists of straight line segments from $\left(\left((n+1)T_{VM_1} - C_{VM_1}\right), nC_{VM_1}\right)$ to $\left((n+1)T_{VM_1}, (n+1)C_{VM_1}\right)$ for n = 0, 1, 2,…. As a result,

$$f(t, T_{VM_1}, C_{VM_1}) = \left\lfloor \frac{t-(T_{VM_1}-C_{VM_1})}{T_{VM_1}} \right\rfloor C_{VM_1} + min\left(\left(t - (T_{VM_1} - C_{VM_1}) - \left\lfloor \frac{t-(T_{VM_1}-C_{VM_1})}{T_{VM_1}} \right\rfloor T_{VM_1}\right), C_{VM_1}\right) \tag{10}$$

With $R_{VM_1} = C_{VM_1}$, since VM_1 has the highest, Figure 5 shows that the (pseudo-)inverse function for VM_1

$$f^{-1}(t, T_{VM_1}, C_{VM_1}) = \left(T_{VM_1} - C_{VM_1}\right) + t + \left(\left\lceil \frac{t}{C_{VM_1}} \right\rceil - 1\right)(T_{VM_1} - C_{VM_1}) \tag{11}$$

When comparing function in Equation (11) with function in Equation (5), i.e., the pessimistic case, we see that the only difference is that $2(T_{VM} - C_{VM})$ in (5) is replaced by $\left(T_{VM_1} - C_{VM_1}\right)$ in (11). The reason for this is that in Equation (11) we know that VM_1 has the highest priority and cannot be blocked by other VMs.

The worst-case response time for task $\tau_{1,j}$ is $r_{1,j} = f^{-1}\left(R_{1,j}, T_{VM_1}, C_{VM_1}\right)$

In order to meet all deadlines for all tasks $\tau_{i,j}$, we need to select T_{VM_1} and C_{VM_1} such that is,

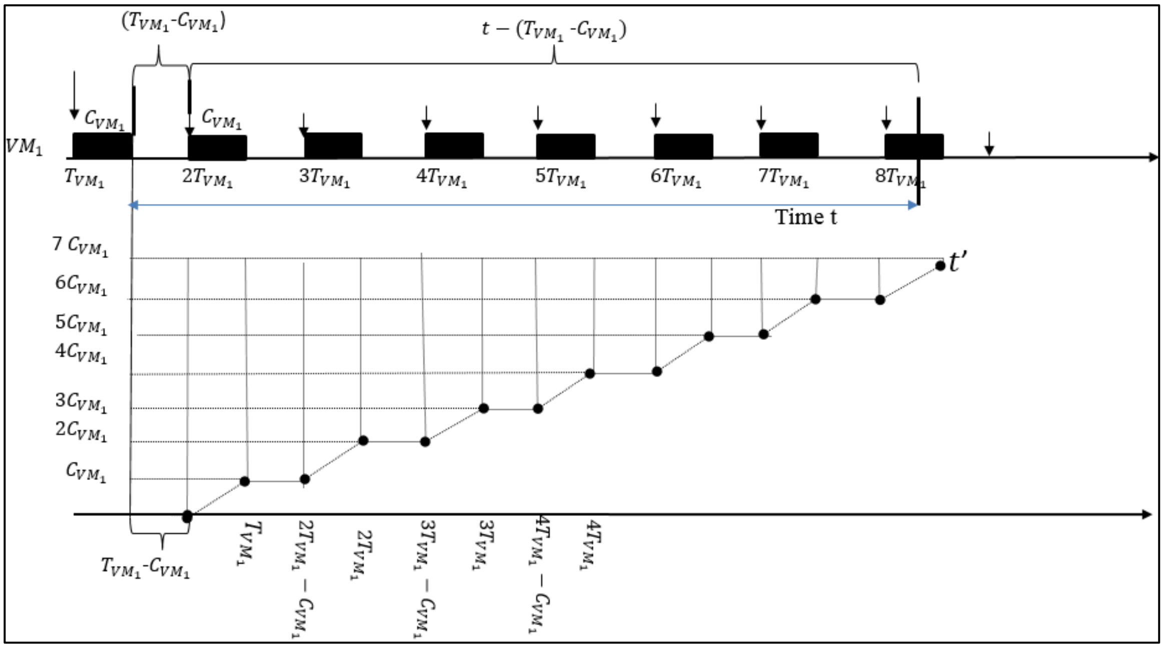


Figure 4: The worst-case scenario for VM_1

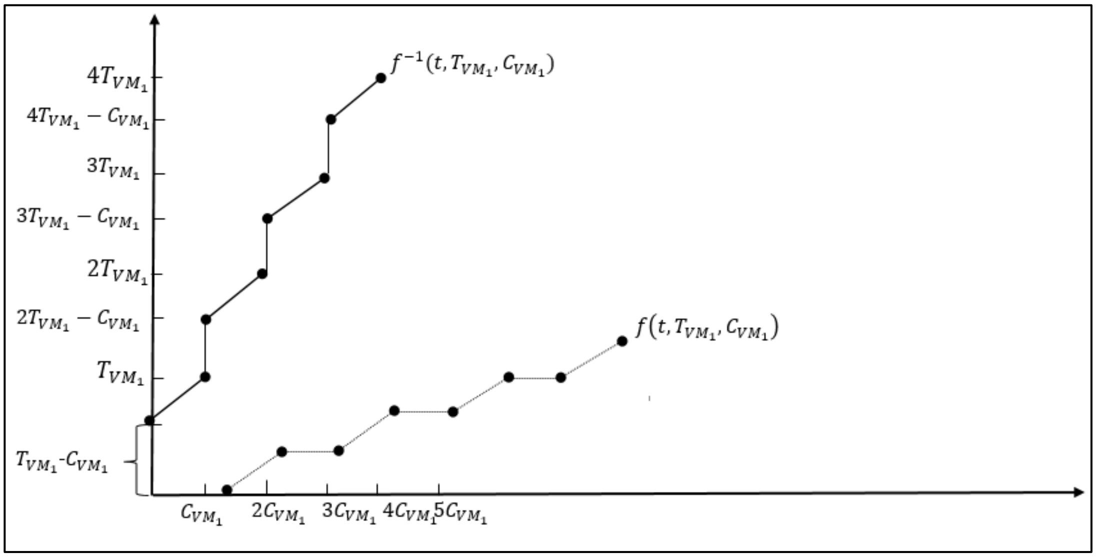


Figure 5: The (pseudo-)inverse function for VM$_1$

$$r_{1,j} = f^{-1}\left(R_{1,j}, T_{VM_1}, C_{VM_1}\right) \leq T_{1,j} \;.$$

VM_2 has the second highest priority and will suffer interference only from VM_1 (see Figure 6). The worst-case response time of VM_2 is given in Equation (12).

$$R_{VM_2} = \; C_{VM_2} + \left\lceil \frac{R_{VM_2}}{T_{VM_1}} \right\rceil C_{VM_1} \qquad (12)$$

In the worst-case VM_2 will wait for $(T_{VM_2} - 2C_{VM_2} + R_{VM_2})$ before it can start its first execution.

The number of whole periods of length T_{VM_2} that is covered by *t* for the worst-case scenario with each period T_{VM_2} contains a total execution C_{VM_2} is $\left\lfloor \frac{t-\left(T_{VM_2}-2C_{VM_2}+R_{VM_2}\right)}{T_{VM_2}} \right\rfloor$.

Let *t'* denotes the minimum amount of time that the VM is running during a time period of length *t*. Then we have

$$t' = max\left(0, \left\lfloor \frac{t-\left(T_{VM_2}-2C_{VM_2}+R_{VM_2}\right)}{T_{VM_2}} \right\rfloor C_{VM_2} + min\left(t - \left(T_{VM_2} - 2C_{VM_2} + R_{VM_2}\right) - \left\lfloor \frac{t-\left(T_{VM_2}-2C_{VM_2}+R_{VM_2}\right)}{T_{VM_2}} \right\rfloor T_{VM_2}, C_{VM_2}\right)\right) \qquad (13)$$

For fixed T_{VM_2}, C_{VM_2} and R_{VM_2}, Figure 6 shows that $t' = f\left(t, T_{VM_2}, C_{VM_2}, R_{VM_2}\right)$ is an increasing function that consists of straight line segments from $\left(\left((n+1)T_{VM_2} - 2C_{VM_2} + R_{VM_2}\right), nC_{VM_2}\right)$ to $\left(\left((n+1)T_{VM_2} - C_{VM_2} + R_{VM_2}\right), (n+1)C_{VM_2}\right)$ for *n* = 0, 1, 2….

As a result,

$$f\left(t, T_{VM_2}, C_{VM_2}, R_{VM_2}\right) = \left\lfloor \frac{t - \left(T_{VM_2} + R_{VM_2} - 2C_{VM_2}\right)}{T_{VM_2}} \right\rfloor C_{VM_2} + min\left(\left(t - \left(T_{VM_2} + R_{VM_2} - 2C_{VM_2}\right) - \left\lfloor \frac{t-\left(T_{VM_2}+R_{VM_2}-2C_{VM_2}\right)}{T_{VM_2}} \right\rfloor T_{VM_2}\right), C_{VM_2}\right) \qquad (14)$$

Figure 7 shows the corresponding (pseudo-)inverse function that is also defined in Equation (15).

$$f^{-1}\left(t, T_{VM_2}, C_{VM_2}, R_{VM_2}\right) = \left(T_{VM_2} - 2C_{VM_2} + R_{VM_2}\right) + t + \left(\left\lceil \frac{t}{C_{VM_2}} \right\rceil - 1\right)\left(T_{VM_2} - C_{VM_2}\right) \quad (15)$$

The worst-case response time for task $\tau_{2,j}$ is

$$r_{2,j} = f^{-1}\left(R_{2,j}, T_{VM_2}, C_{VM_2}, R_{VM_2}\right) \qquad (16)$$

In order to meet all deadlines for all tasks $\tau_{i,j}$ we must select T_{VM_2} and C_{VM_2} such that

$$r_{2,j} = f^{-1}\left(R_{2,j}, T_{VM_2}, C_{VM_2}, R_{VM_2}\right) \leq \; T_{2,j} \qquad (17)$$

Figure 8 shows that in general for the worst-case scenario, *t* starts with a $T_{VM_i} + R_{VM_i} - 2C_{VM_i}$ period before VM_i can start its execution (R_{VM_i} is the worst-case response time of VM_i).

The number of complete periods of length T_{VM_i}, with execution C_{VM_i}, that are covered by *t* for the worst-case scenario is given by Equation (18).

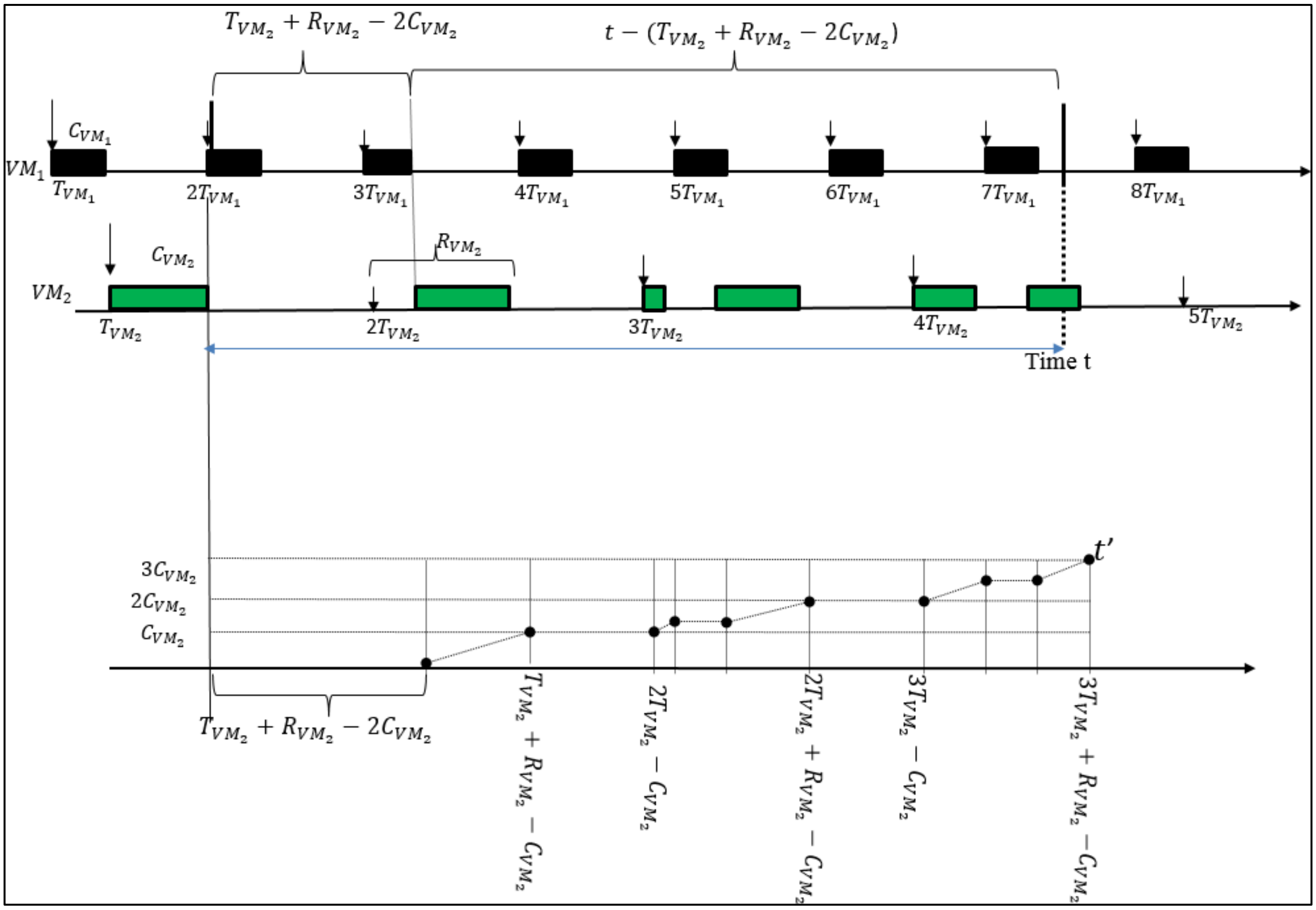


Figure 6: The worst-case scenario for VM_2.

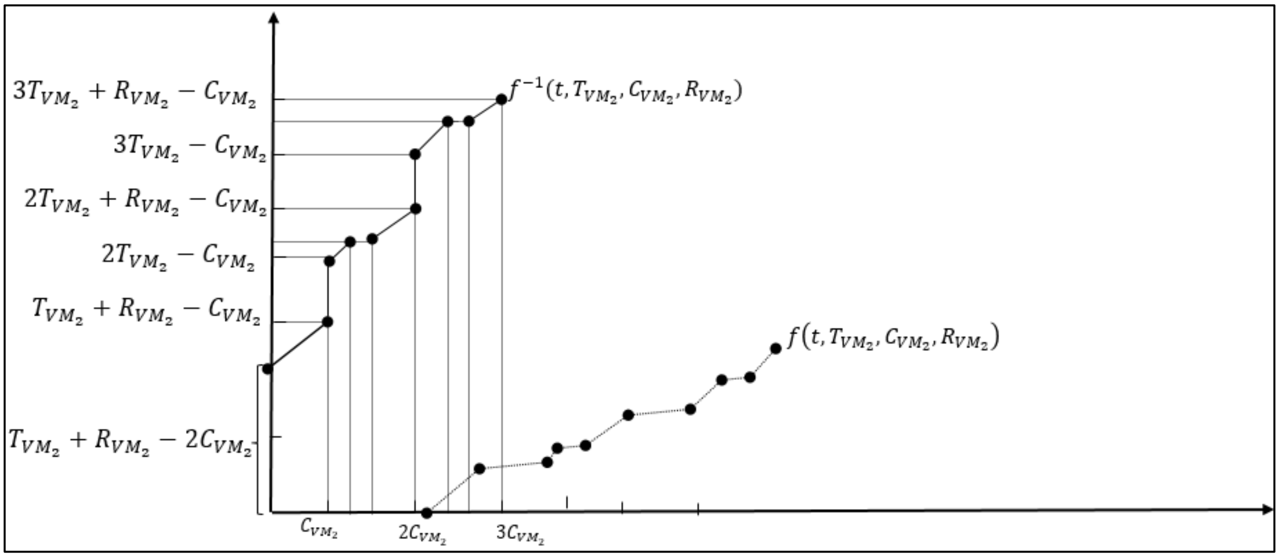


Figure 7: The (pseudo-)inverse function for VM_2

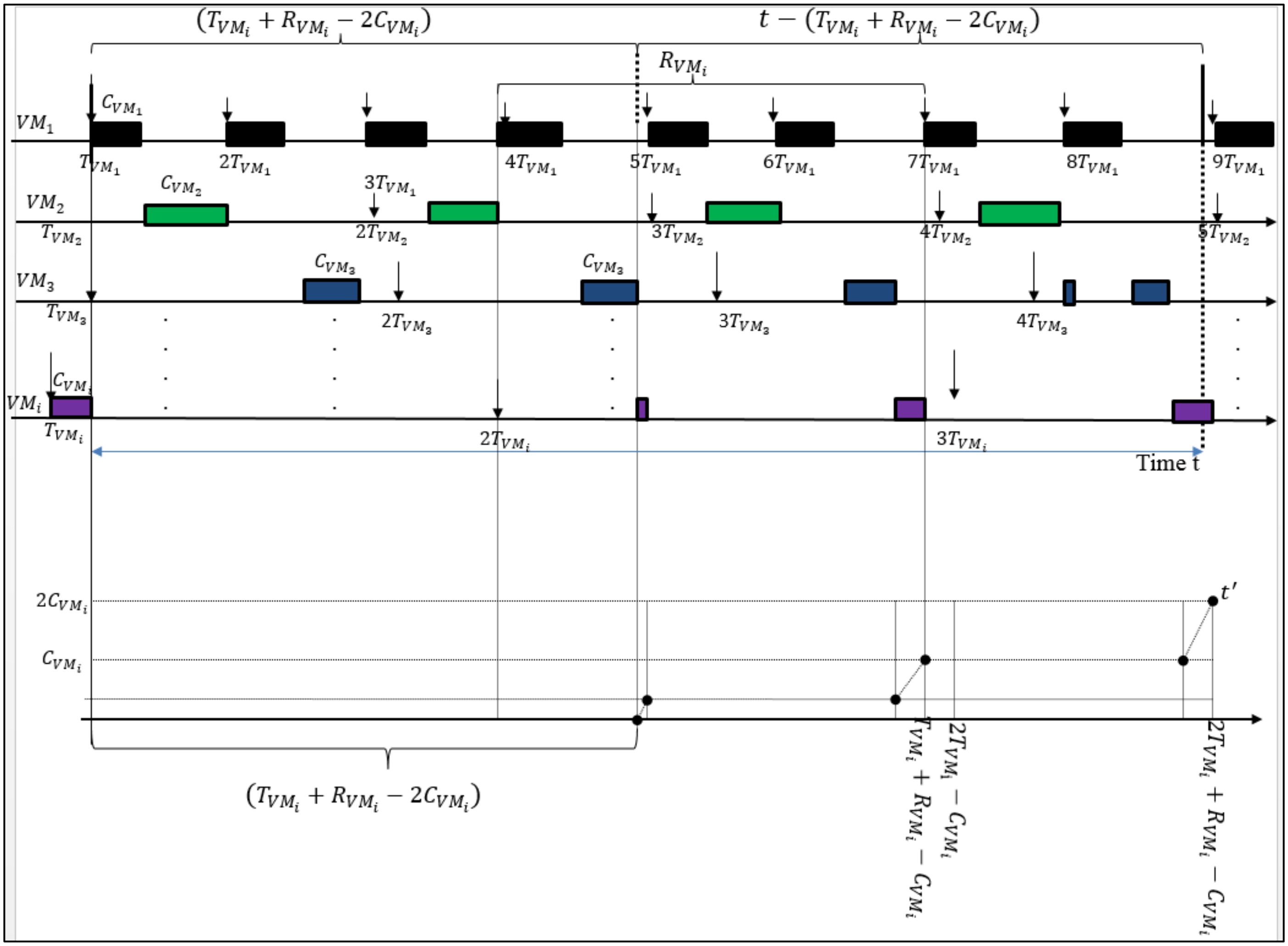


Figure 8: The worst-case scenario for $\boldsymbol{VM_i}$

$$\left\lfloor \frac{t-(T_{VM_i}-2C_{VM_i}+R_{VM_i})}{T_{VM_i}} \right\rfloor \tag{18}$$

Let *t'* denotes the minimum time that the VM is running during a time period of length *t*. Then we have

$$t' = max\left(0, \left\lfloor \frac{t-\left(T_{VM_i}-2C_{VM_i}+R_{VM_i}\right)}{T_{VM_i}} \right\rfloor C_{VM_i} + min\left(\left(\left(t - (T_{VM_i} - 2C_{VM_i} + R_{VM_i}) - \left\lfloor \frac{t-\left(T_{VM_i}-2C_{VM_i}+R_{VM_i}\right)}{T_{VM_i}} \right\rfloor T_{VM_i}\right), C_{VM_i}\right)\right)\right) \tag{19}$$

For fixed T_{VM_i}, C_{VM_i} and R_{VM_i}, Figure 8 shows that $t' = f(t, T_{VM_2}, C_{VM_2}, R_{VM_2})$ is an increasing function that consists of straight line segments from$\left(\left((n+1)T_{VM_i} - 2C_{VM_i} + R_{VM_i}\right), nC_{VM_i}\right)$ to $\left(\left((n+1)T_{VM_i} - C_{VM_i} + R_{VM_i}\right), (n+1)C_{VM_i}\right)$ for n = 0, 1, 2…., as a result,

$$f(t, T_{VM_i}, C_{VM_i}, R_{VM_i}) = \left\lfloor \frac{t-(T_{VM_i}+R_{VM_i}-2C_{VM_i})}{T_{VM_i}} \right\rfloor C_{VM_i} + min\left(\left(\left(t - (T_{VM_i} + R_{VM_i} - 2C_{VM_i})\right) - \left\lfloor \frac{t-(T_{VM_i}+R_{VM_i}-2C_{VM_i})}{T_{VM_i}} \right\rfloor T_{VM_i}\right), C_{VM_i}\right) \tag{20}$$

The number of complete periods of length T_{VM_i}, with execution C_{VM_i}, that are covered by *t* for the worst-case scenario is shown in Figure 9, and Equation (21) shows the general inverse function that maps virtual time to the worst-case real-time for VM_i.

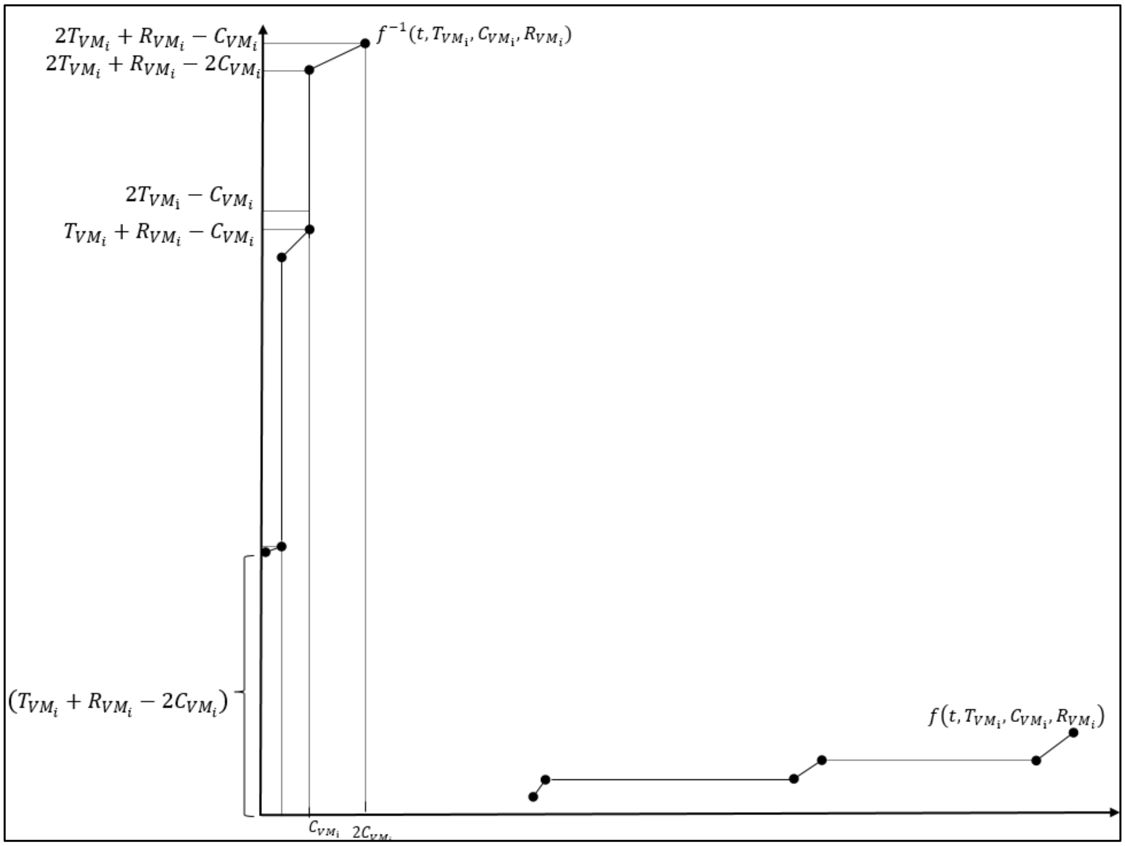

$2T_{VM_i} + R_{VM_i} - C_{VM_i}$
$2T_{VM_i} + R_{VM_i} - 2C_{VM_i}$
$f^{-1}(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$
$2T_{VM_i} - C_{VM_i}$
$T_{VM_i} + R_{VM_i} - C_{VM_i}$
$(T_{VM_i} + R_{VM_i} - 2C_{VM_i})$
$f(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$
C_{VM_i}
$2C_{VM_i}$

Figure 9: The (pseudo-)inverse function for $\boldsymbol{VM_i}$

$$f^{-1}(t, T_{VM_i}, C_{VM_i}, R_{VM_i}) = (T_{VM_i} - 2C_{VM_i} + R_{VM_i}) + t + \left(\left\lceil \frac{t}{C_{VM_i}} \right\rceil - 1\right)(T_{VM_i} - C_{VM_i}) \quad (21)$$

with $0 \leq i \leq k$ and $R_{VM_i} = C_{VM_i} + \sum_{m=1}^{i-1} \left\lceil \frac{R_{VM_i}}{T_{VM_m}} \right\rceil C_{VM_m}$

The worst-case response time $r_{i,j}$ for task $\tau_{i,j}$ is ($R_{i,j}$ is defined in Equation (1)) $r_{i,j} = f^{-1}(R_{i,j}, T_{VM_i}, C_{VM_i}, R_{VM_i})$.

The function in Equation (21) becomes equal to the pessimistic case when $R_{VM_i} = T_{VM_i}$

5 Accounting for Overhead

Whenever a preemption takes place, different sources of overhead should be taken into account. Previous studies have considered overhead in compositional real-time systems [5, 23]. There are two important differences between these studies and our study:

First, in the previous studies the authors did not assume that we have information about the entire work-load i.e., they assumed the pessimistic approach.

Second, in compositional real-time systems the components are abstractions and do not correspond to any execution time entity such as a VM. In our approach we inflate the execution time of each VM to compensate for context switching overhead between VMs (see Figure 10).

Overhead due to context switching between tasks inside a VM is orthogonal to our approach and can be handled in the same way as in non-virtualized systems, e.g., by inflating the execution times of the real-time tasks.

In every execution cycle, the VM worst-case execution time is inflated by an X which is an accumulation of cache overhead, release overhead, and some other overheads that are part of a context switch. The maximum number of preemptions suffered by a given VM is bounded by the number of releases of higher priority VMs within its response time R_{VM}, e.g., in Figure 10, VM_i with $i = 3$, has $4X$ overhead, $1X$ is the initial startup overhead of the VM_3, $2X$ preemptions from VM_1 and $1X$ from VM_2 in the worst-case scenario. Figure 11 shows the pseudo-inverse function for VM_i with overhead. The inflated execution time is given by Equation (22)

$$C'_{VM_i} = C_{VM_i} + X + \sum_{k=1}^{i-1} \left(\left\lceil \frac{R'_{VM_i}}{T_{VM_k}} \right\rceil X \right) \quad (22)$$

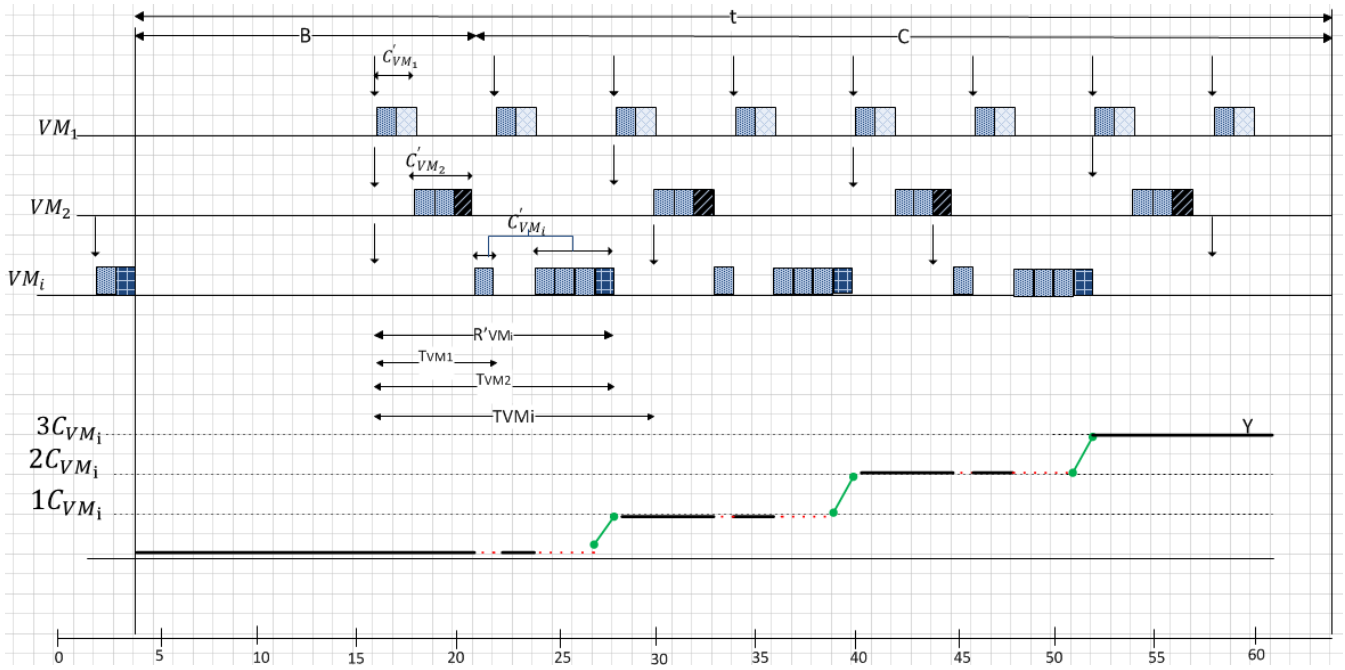
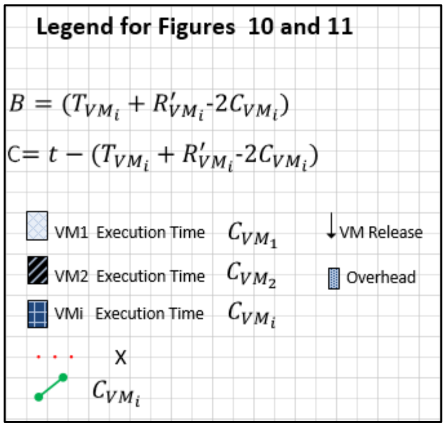


Figure 10: The worst-case scenario for VM_i with overhead


Legend for Figures 10 and 11
$B = (T_{VM_i} + R'_{VM_i}\text{-}2C_{VM_i})$
$C = t - (T_{VM_i} + R'_{VM_i}\text{-}2C_{VM_i})$
VM1 Execution Time C_{VM_1}
VM Release
VM2 Execution Time C_{VM_2}
Overhead
VMi Execution Time C_{VM_i}
X
C_{VM_i}

And the inflated response time is given by Equation (23)

$$R'_{VM_i} = C'_{VM_i} + \sum_{j=1}^{i-1}\left(\left\lceil\frac{R'_{VM_i}}{T_{VM_j}}\right\rceil C'_{VM_j}\right) \qquad (23)$$

Equations (22) and (23) are solved using numeric iterative method, Figure 12 describes this algorithm, below is the description of Figure 12.

1. Inflated execution time is initialized, i.e. $C'_{VM_i} = C_{VM_i} + iX$
2. Inflated response time R'_{VM_i} is calculated using initial inflated execution time.
3. Inflated execution time C'_{VM_i} is calculated, this C'_{VM_i} is again used to calculate R'_{VM_i}, this step iterates until R'_{VM_i} value does not change anymore.
4. If R'_{VM_i} value does not change anymore, then we get the value for R'_{VM_i}, and for C'_{VM_i}.

   E.g., in Figure 10, if $i = 3$, $C_{VM_1} = 1$, $C_{VM_2} = 1$, $C_{VM_3} = 1$, $T_{VM_1} = 6$, $T_{VM_2} = 12$, $T_{VM_3} = 14$, and $X = 1$, $C'_{VM_1} = C_{VM_1} + X = 2$, since VM_1 has the highest priority, $R'_{VM_1} = 2$.

For VM_2 we have,

$C'_{VM_2} = C_{VM_2} + 2X = 3$, $R'_{VM_2} = 3 + \left\lceil\frac{3}{6}\right\rceil 2 = 5$, $C'_{VM_2} = 2 + \left\lceil\frac{5}{6}\right\rceil 1 = 3$, $R'_{VM_2} = 3 + \left\lceil\frac{5}{6}\right\rceil 2 = 5$,

Therefore $R'_{VM_2} = 5, C'_{VM_2} = 3$

For VM_3 we have,

$C'_{VM_3} = C_{VM_3} + 3X = 4$, $R'_{VM_3} = 4 + \left\lceil\frac{4}{6}\right\rceil 2 + \left\lceil\frac{4}{12}\right\rceil 3 = 9$, $C'_{VM_3} = 2 + \left\lceil\frac{9}{6}\right\rceil 1 + \left\lceil\frac{9}{12}\right\rceil 1 = 5$

$R'_{VM_3} = 5 + \left\lceil\frac{9}{6}\right\rceil 2 + \left\lceil\frac{9}{12}\right\rceil 3 = 12$, $C'_{VM_3} = 2 + \left\lceil\frac{12}{6}\right\rceil 1 + \left\lceil\frac{12}{12}\right\rceil 1 = 5$, $R'_{VM_3} = 5 + \left\lceil\frac{12}{6}\right\rceil 2 + \left\lceil\frac{12}{12}\right\rceil 3 = 12$

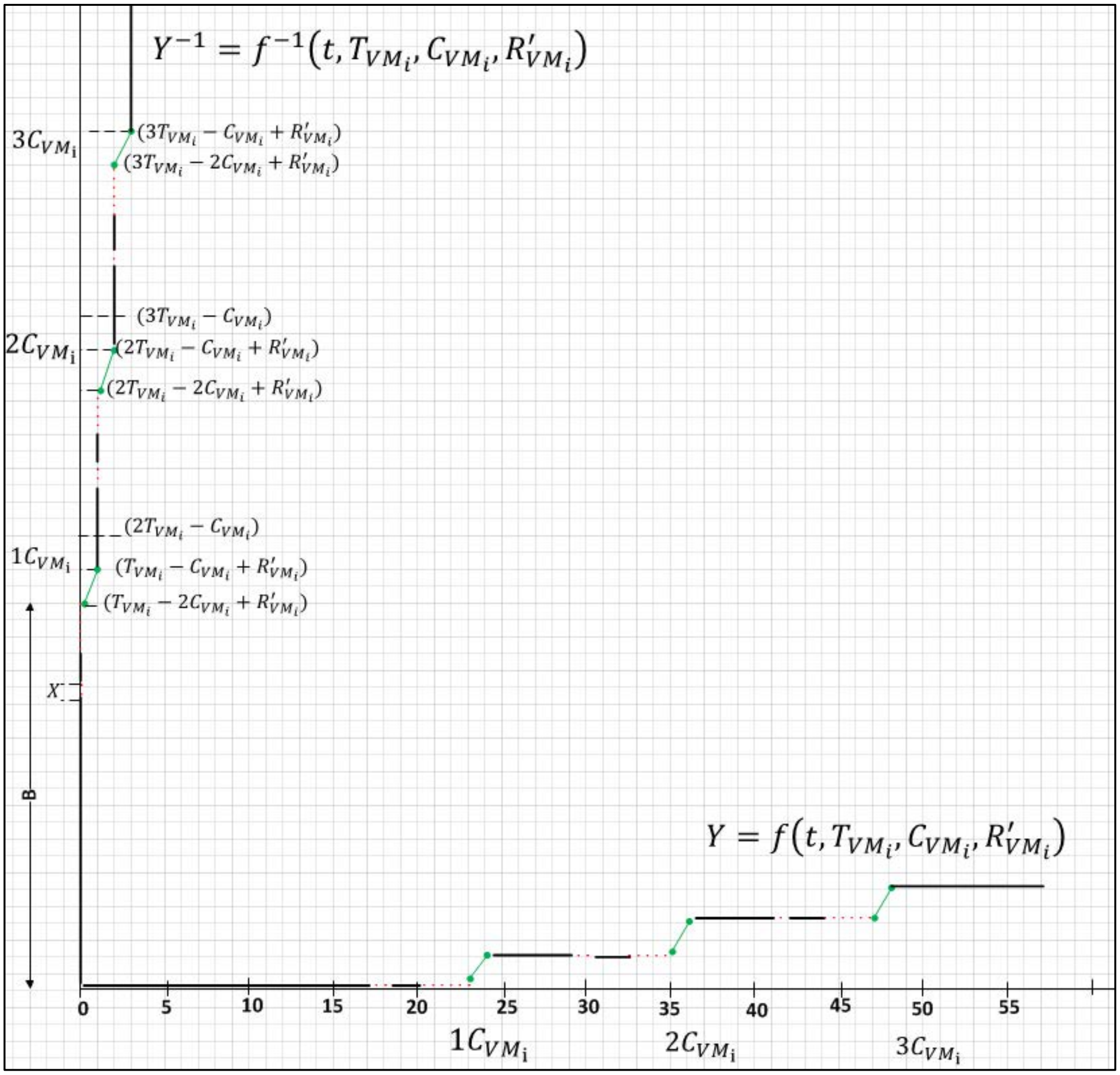


Figure 11: The (pseudo-)inverse function for VM_i with overhead.

Therefore $R'_{VM_3} = 12$, $C'_{VM_3} = 5$

In the optimistic case, while considering the overhead, the pseudo-inverse function to calculate the worst-case response time is given by Equation (24)

$$f^{-1}(t, T_{VM_i}, C_{VM_i}, R'_{VM_i}) = (T_{VM_i} - 2C_{VM_i} + R'_{VM_i}) + t + \left(\left\lceil \frac{t}{C_{VM_i}} \right\rceil - 1\right)(T_{VM_i} - C_{VM_i}) \quad (24)$$

Let C'_{VM_j} denotes the inflated worst-case execution time for VM_j. In that case we make a pessimistic but safe assumption by accumulating all the overhead at the beginning of every execution cycle. This gives a straight forward way of estimating the maximum overhead that a VM will face in every period e.g. if VM_i is released there is an overhead represented by one X, and it is preempted 2 times by VM_2 and 2 times by VM_1, in Figure 10, all X values will be added up, hence the total overhead is 5X.

Figure 11 shows the minimum time *t* that VM_i can run. The worst-case is when *t* starts right after VM_i has finished an execution period that started directly after a release of VM_i. The worst-case execution time C_{VM} of higher priority VMs becomes C'_{VM} after overhead inflation. In order to find the worst-case response time of VM_i during time *t* we consider a case when all VMs are released at the same time.

6 Example when Overhead is Omitted

Tables 1 and 2 contain details about two programs: program one is executed in virtual machine one (VM_1) and program two is executed in virtual machine two (VM_2); each program has 3 tasks.

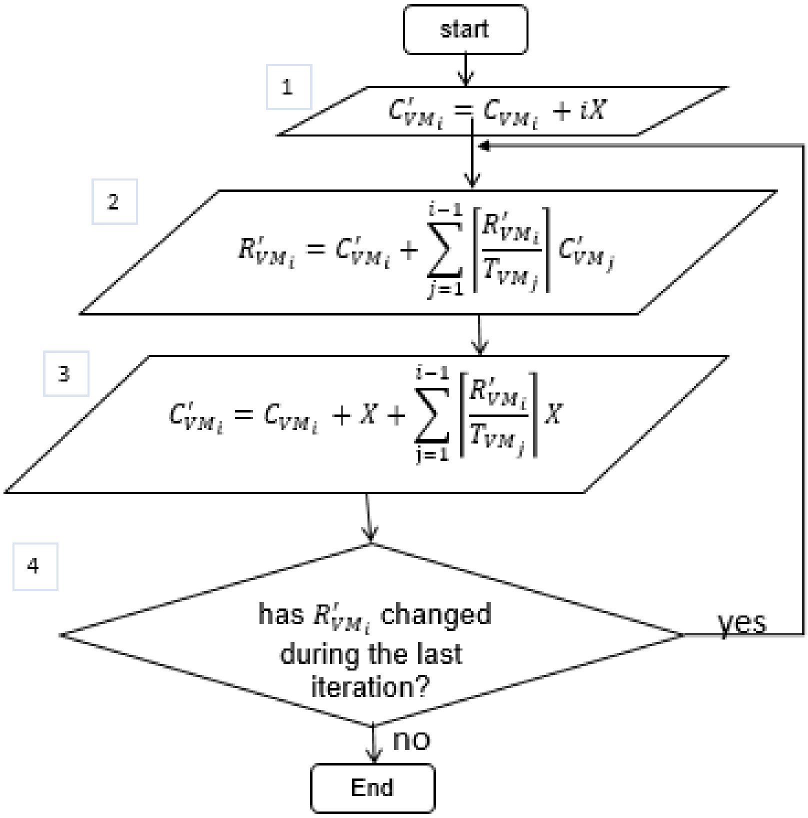

start
1
$C'_{VM_i} = C_{VM_i} + iX$
2
$R'_{VM_i} = C'_{VM_i} + \sum_{j=1}^{i-1} \left\lceil \frac{R'_{VM_i}}{T_{VM_j}} \right\rceil C'_{VM_j}$
3
$C'_{VM_i} = C_{VM_i} + X + \sum_{j=1}^{i-1} \left\lceil \frac{R'_{VM_i}}{T_{VM_j}} \right\rceil X$
4
has R'_{VM_i} changed during the last iteration?
yes
no
End

Figure 12: The algorithm flow chart to find $\boldsymbol{C'_{VM_i}}$ and $\boldsymbol{R'_{VM_i}}$.

Table 1: Program one with 3 tasks executing in VM_1

Task	**Period ($T_{i,j}$)**	**Worst-case execution time ($C_{i,j}$)**	**Utilization ($U_{i,j}$)**
$\tau_{1,1}$	16	2	2/16 = 0.125
$\tau_{1,2}$	24	1	1/24 = 0.042
$\tau_{1,3}$	36	4	4/36 = 0.111
Total Utilization			**0.278**

Table 2: Program two with 3 tasks executing in VM_2

Task	**Period ($T_{i,j}$)**	**Worst-case execution time ($C_{i,j}$)**	**Utilization ($U_{i,j}$)**
$\tau_{2,1}$	28	1	1/28 = 0.035
$\tau_{2,2}$	34	1	1/34 = 0.029
$\tau_{2,3}$	38	2	2/38 = 0.052
Total Utilization			**0.116**

First, we use the pessimistic method where we look at each VM in isolation. Let us assume that VM_1 uses 40%, and VM_2 uses 30%, of the CPU, i.e. $\mathrm{C}_{\mathrm{VM}_1}/\mathrm{T}_{\mathrm{VM}_1} = 0.4$ and $\mathrm{C}_{\mathrm{VM}_1}/\mathrm{T}_{\mathrm{VM}_1} = 0.3$.

When $C_{VM_1}/T_{VM_1} = 0.4$, we can replace C_{VM_1} by $0.4\,T_{VM_1}$ in Equation (5), thus obtaining the function $f^{-1}\left(t, T_{VM_1}, 0.4T_{VM_1}\right) = 1.2T_{VM_1} + t + \left(\left\lceil \frac{t}{0.4T_{VM_1}} \right\rceil - 1\right) 0.6T_{VM_1}$

We start by looking at VM_1 and task $\tau_{1,1}$. We want to find the maximum T_{VM_1} such that;

$$r_{1,1} = f^{-1}\left(R_{1,1}, T_{VM_1}, 0.4T_{VM_1}\right) \leq T_{1,1}$$

$R_{1,1} = C_{1,1}$, since $\tau_{1,1}$ has the highest priority in program one. Therefore, we get the equation

$$1.2\,T_{VM_1} + 2 + \left(\left\lceil \frac{2}{0.4T_{VM_1}} \right\rceil - 1\right) 0.6T_{VM_1} = 16,$$

which gives $T_{VM_1} = 11.7$ and $C_{VM_1} = 0.4 * 11.7 = 4.68$.

The first execution period will, in the worst-case, start at time $2(T_{VM_1} - C_{VM_1}) = 1.2T_{VM_1} = 14$. Since $T_{1,1} = 16$ and $C_{1,1} = 2$ we see that $\tau_{1,1}$ will execute two times back-to-back in this interval, i.e., after the first execution of $\tau_{1,1}$ it will be released again at time 16. Consequently, $\tau_{1,2}$ cannot start executing until time 18. The first execution period of VM_1 will end at $2T_{VM_1} - C_{VM_1} = 1.6\,T_{VM_1} = 1.6 * 11.7 = 18.7$. Since $C_{1,2} = 1$, it cannot complete during the first period of VM_1. The second period of VM_1 starts at $3T_{VM_1} - 2C_{VM_1} = 2.2\,T_{VM_1} = 2.2 * 11.7 = 25.7$ which is after the deadline of $\tau_{1,2}$ ($T_{1,2} = 24$). Task $\tau_{1,3}$ will also miss its deadline. In [2] the authors looked at this task set and found that 10.8 is the largest period that $\tau_{1,2}$ can tolerate and that 10 is the largest period that $\tau_{1,3}$ can tolerate, when $C_{VM_1}/T_{VM_1} = 0.4$. This means that the maximum period T_{VM_1} that will guarantee that all three tasks will meet their deadlines is 10 when $C_{VM_1}/T_{VM_1} = 0.4$.

For VM_2, with $C_{VM_2} = 0.3T_{VM_2}$, the inverse function in Equation (5) becomes

$$f^{-1}\left(t, T_{VM_2}, 0.3T_{VM_2}\right) = 1.4T_{VM_2} + t + \left(\left\lceil \frac{t}{0.3T_{VM_2}} \right\rceil - 1\right) 0.7T_{VM_2}$$

By using the method above, we get $T_{VM_2} = 19.28$, and thus $C_{VM_2} = 0.3 * 19.28 = 5.78$ which makes all tasks in VM_2 meet their deadline when $C_{VM_1}/T_{VM_1} = 0.3$.

We now use our method where we consider all VMs. We are again using fixed priorities, and the examples in tables 1 and 2. Program one is scheduled on VM_1, and we now know that VM_1 has the highest priority. Therefore, we use the inverse function in Equation (11) to calculate the maximum T_{VM_1} such that we know that task $\tau_{1,1}$ meets its deadline.

$$f^{-1}\left(t, T_{VM_1}, 0.4T_{VM_1}\right) = 0.6T_{VM_1} + t + \left(\left\lceil \frac{t}{0.4T_{VM_1}} \right\rceil - 1\right) 0.6T_{VM_1}$$

We want to find the maximal T_{VM_1} such that $r_{1,1} = f^{-1}\left(R_{1,1}, T_{VM_1}, 0.4T_{VM_1}\right) \leq T_{1,1}$.

Note that $R_{1,1} = C_{1,1}$, since $\tau_{1,1}$ has the highest priority in program one.

From this we get $0.6T_{VM_1} + 2 + \left(\left\lceil \frac{2}{0.4T_{VM_1}} \right\rceil - 1\right) 0.6T_{VM_1} = 16$.

From this we get $T_{VM_1} = 23.33$. If we have a period of 23.33 we get $C_{VM_1} = 0.4 * 23.33 = 9.33$. The first execution period C_{VM_1} will in the worst-case start at time at $T_{VM_1} - C_{VM_1} = 14$ and it will end at $T_{VM_1} = 23.33$. Since $\mathrm{T}_{1,1} = 16$ and $\mathrm{C}_{1,1} = 2$ we see that $\tau_{1,1}$ will execute two times back-to-back in this interval, i.e., after the first execution of $\tau_{1,1}$ from 14 to 16, and task $\tau_{1,1}$ will be released again at time 16 and execute from 16 to 18. Since $\mathrm{C}_{1,2} = 1$ and $\tau_{1,2} = 24$, $\tau_{1,2}$ will execute from 18 to 19. It is clear that $\tau_{1,3}$ will also execute in the first cycle from 19 to 23, since $\mathrm{C}_{1,3} = 4$. It is thus clear that all tasks in VM_2 will meet their deadlines for $T_{VM_1} = 23.33$ when $C_{VM_1}/T_{VM_1} = 0.4$.

We now do the same thing for VM_2. VM_2 has the second highest priority, and may be interrupted by VM_1. We use the inverse function in Equation (15), and $C_{VM_2} = 0.3T_{VM_2}$. The worst-case response time (see Equation (17)) $r_{2,1} = f^{-1}\left(R_{2,1}, T_{VM_2}, C_{VM_2}, R_{VM_2}\right) \leq T_{2,1} = 28$ (N.B. $R_{2,1} = C_{2,1}$).

With $R_{VM_2} = C_{VM_2} + \left\lceil \frac{R_{VM_2}}{T_{VM_1}} \right\rceil C_{VM_1}$, and $C_{VM_2} = 0.3T_{VM_2}$ we get $R_{VM_2} = 0.3T_{VM_2} + \left\lceil \frac{R_{VM_2}}{23.33} \right\rceil 9.33$. By using our formulas we see that $r_{2,1} = 28$ for $T_{VM_2} = 25.24$, i.e., this is the maximum period for VM_2 that task $\tau_{2,1}$ can tolerate, $C_{VM_2} = 7.57$, and $R_{VM_2} = 16.9$. The first execution period C_{VM_2} will in the worst-case start at time $T_{VM_2} - 2C_{VM_2} + R_{VM_2} = 27$ and it will end at $T_{VM_2} - C_{VM_2} + R_{VM_2} = 34.57$. Since $T_{2,1} = 28$ and $C_{2,1} = 1$ we see that $\tau_{2,1}$ will execute two times back-to-back in this interval, i.e., after the first execution of $\tau_{2,1}$ from 27 to 28, it will be released again at time 28 and executes from 28 to 29. Since $C_{2,2} = 1$ and $T_{2,2} = 34$, $\tau_{2,2}$ will execute from 29 to 30. It is clear that $\tau_{2,3}$ will also execute in the first cycle from 30 to 32 since $C_{2,3} = 2$ and $T_{2,3} = 38$. It is thus clear that all tasks in VM_2 will meet their deadlines. As shown in the example, the method that looks at all VMs, gives longer periods than the method that takes each VM individually, e.g., T_{VM_1} increases from 10 to 23.33 and T_{VM_2} increases from 19.28 to 25.24.

Figures 13(a), (b), and (c) show the values that T_{VM} can take with respect to different values of C_{VM_i}/T_{VM_i}. Pessimistic means that we treat each VM in isolation, whereas optimistic means that we considered all VMs. The detailed example above is extended below; we calculate the maximal T_{VM_i} for each of the n_i tasks in the program, such that $r_{i,j} \leq T_{i,j}$ for $C_{VM_i} = uT_{VM_i}$ (u is the C_{VM_i}/T_{VM_i} value that we consider, i.e., the values on the

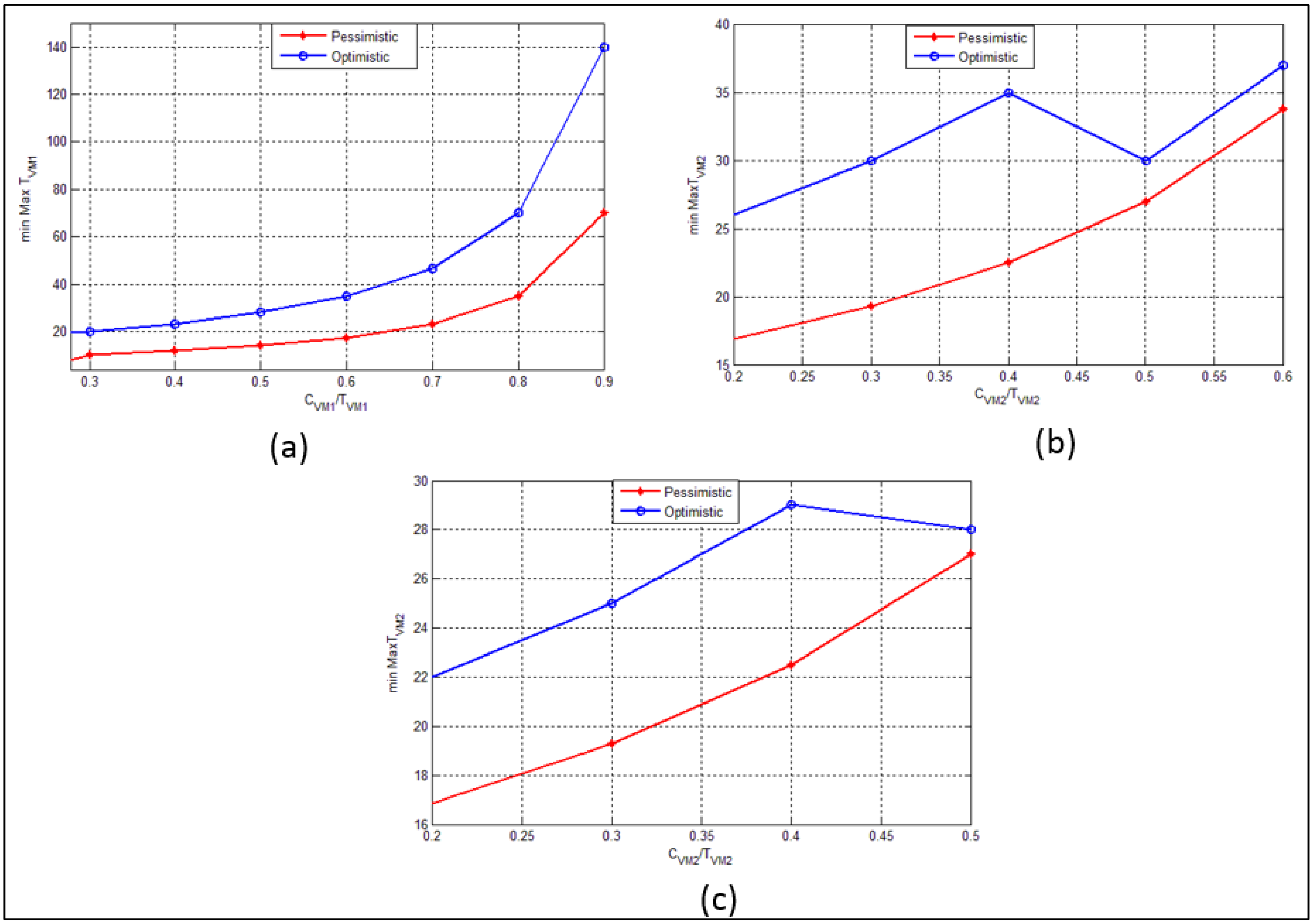


Figure 13(a): C_{VM_1}/T_{VM_1} versus minimum of the max T_{VM_1}, (b) C_{VM_2}/T_{VM_2} versus minimum of the max T_{VM_2} when $C_{VM_1}/T_{VM_1} = 0.3$, **(c)** C_{VM_2}/T_{VM_2} versus minimum of the max T_{VM_2} when $C_{VM_1}/T_{VM_1} = 0.4$.

x-axis) and then we choose the shortest of these n_i values on T_{VM_i}. We call this value the minimum of the maximum (min max) T_{VM_i}. Figure 13 (a) shows (min max) T_{VM_1} as a function of C_{VM_1}/T_{VM_1}, the detailed example above corresponds to the value 0.4 on the *x*-axis.

If we consider the case where we have knowledge about one VM only, we have to make a pessimistic assumption, we see that T_{VM_1} is shorter than the case when we have knowledge of all VMs, i,e., the optimistic case.

Figures 13(b) and (c) plot T_{VM_2} versus C_{VM_2}/T_{VM_2} for $C_{VM_1}/T_{VM_1} = 0.3$ and $C_{VM_1}/T_{VM_1} = 0.4$, respectively, (the pessimistic values are not affected by C_{VM_1}/T_{VM_1}, and are thus the same in both Figures). The detailed example above corresponds to the value 0.3 on the *x*-axis in Figure 13(c). The resource allocated to VM_2 should be less or equal to $1 - C_{VM_1}/T_{VM_1}$, since $(C_{VM_1}/T_{VM_1}) + (C_{VM_2}/T_{VM_2})$ cannot exceed 1.

It may seem counter intuitive that the period T_{VM_2} of the optimistic case may decrease when we increase C_{VM_2}/T_{VM_2}. The reason for this is that when we increase C_{VM_2}/T_{VM_2}, then C_{VM_2} also increases, and this affects R_{VM_2}. When C_{VM_2}/T_{VM_2} goes from 0.4 to 0.5, then R_{VM_2} will increase since C_{VM_1} will interfere two times since $R_{VM_2} \geq T_{VM_1}$ (remember $R_{VM_2} = C_{VM_2} + \left\lceil \frac{R_{VM_2}}{T_{VM_1}} \right\rceil C_{VM_1}$).

In the optimistic case we use the (safe but actually somewhat pessimistic) assumption that in the worst-case VM_2 may not start running until $R_{VM_2} - C_{VM_2}$ time units after the release. To compensate for the double interference from C_{VM_1}, T_{VM_2} decreases when C_{VM_2}/T_{VM_2} increases from 0.4 to 0.5 in Figure 13 (b). The drops of T_{VM_2} in Figure 13 (c) are due to the same effect.

7 Simulation

The scheduling of tasks inside each VM uses RMS. We consider 8 programs that run in one VM each. Each program is a task set of 10 tasks. Tasks periods are randomly generated with a uniform distribution. We assume that the average task periods are not the same in all programs, and we generated random periods for the intervals [200, 800], [300, 900], [400, 1000],…, [900, 1500] for tasks inside $VM_1, VM_2, \ldots, VM_8$ respectively. We see that task periods overlap and we sorted the VMs in increasing average period order. Inspired by the well-known RMS algorithm, we decided to use the average periods as the basis for assigning static priorities to VMs, i.e. VM_1 has the highest priority and VM_8 has the lowest priority. We simulated four cases, case 1 with one VM, case 2 with two VMs, case 3 with four VMs and case 4 with eight VMs.

Each case is simulated for different total utilizations U [0.1, 0.2, …, 0.8]; the utilization is for the entire set of VMs in the simulation, and each VM has the same utilization. For example, if we have two VMs and a utilization of 0.6, then VMs equally share this utilization, i.e., each VM has a total utilization of 0.3. When we have total utilization of a VM, we distribute this total utilization to the 10 tasks inside that VM using the UniFast algorithm [3]. Each task's execution $C_{i,j}$ is then obtained by multiplying the utilization of the task with the task's period.

Each VM will get a period that is half of the shortest period of any task in that VM, which seems to be a reasonable heuristic. The hypervisor will assign a priority to each VM based on the VM period length, the shorter the length the higher the priority. Thereafter, it will find a C_{VM} that makes the task set schedulable using Equation (5) for the pessimistic case, and Equation (21) for the optimistic case, or Equation (24) if overhead is included. The simulation is done for the pessimistic and the optimistic methods. We repeated each unique case 20 times to be able to calculate average values and standard deviations, e.g., one unique case is 4 VMs and a total utilization of 0.4, another one unique case is 4 VMs and a total utilization of 0.6. We used the MATLAB scheduling toolbox TORSCHE (Time Optimization of Resources, SCHEduling) in our simulations [11, 28].

We repeated the simulation for different overhead values (X values) X= [0,1,2,…,9], zero means the absence of overhead and the results are presented in Figure 15. Figures 14(a,b,c,d) show the C_{VM}/T_{VM} versus total task utilization and the standard deviation of C_{VM}/T_{VM}. The figures show that total C_{VM}/T_{VM} increases as the number of VMs increases. These figures also show that the optimistic method performs better than the pessimistic method for all cases. Figure 15 shows that the impact of overhead becomes larger when there are more VMs, e.g., the slope in the overhead weight direction of the planes in Figure 15 is larger for the cases with 4 and 8 VMs compared to the cases with only 1 or 2 VMs. The reason for this is that low priority VMs will suffer from more VM context switches compared to high priority VMs (see Figure 10).

8 Conclusions

In this paper we have extended previous results on two-level (sometimes called hierarchical) scheduling of virtual machines (VMs). Previous studies have considered each VM in isolation. Our contribution is that we have taken a holistic approach and included the entire work-load consisting of k VMs in our method. A simulation study shows that our approach makes it possible to guarantee real-time response requirements for more cases (higher loads) compared to the previous approach (where each VM is analyzed in isolation).

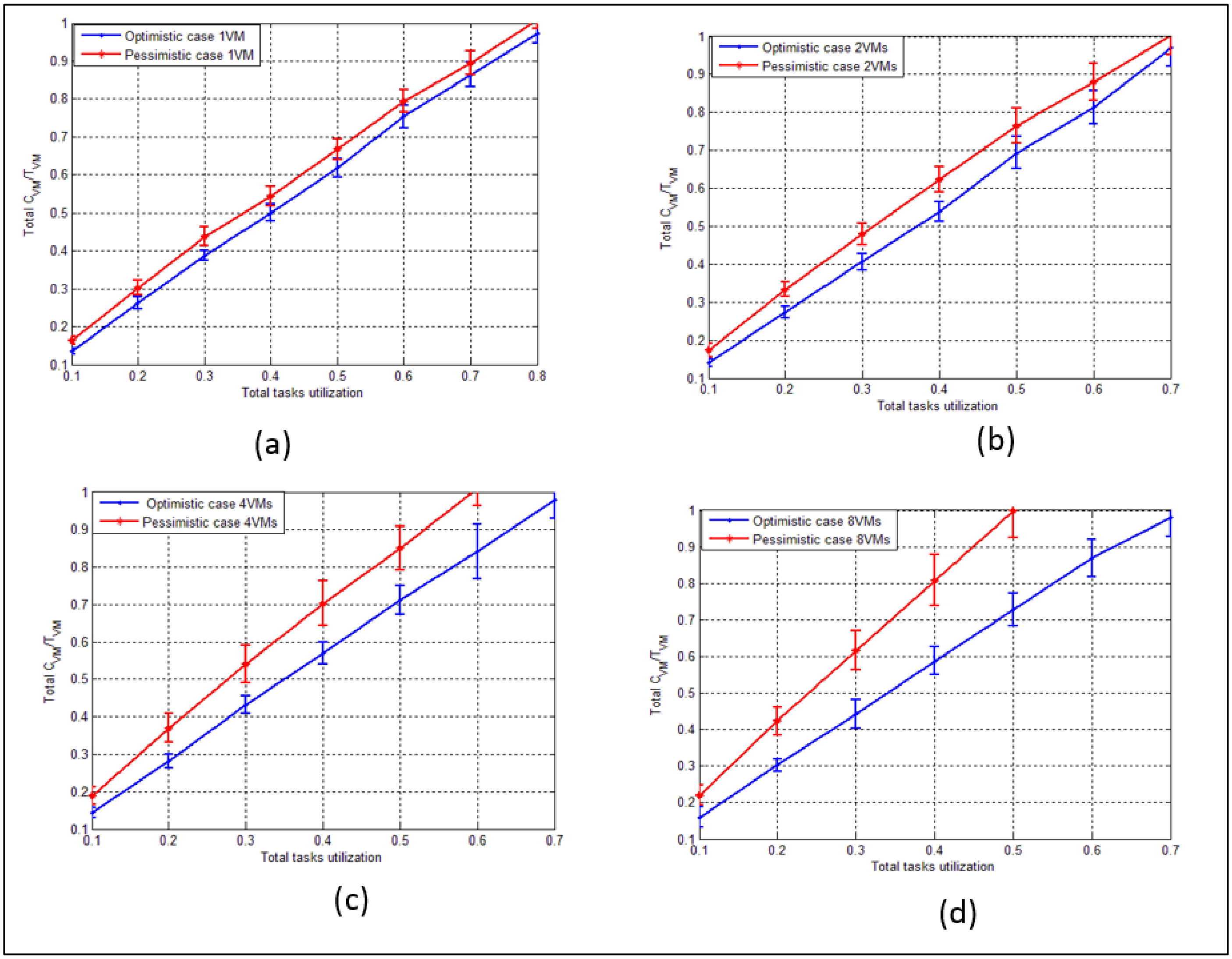
Whether the overhead is accounted for or ignored; we have defined a function $f^{-1}(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$ that returns the maximum (worst-case) wall clock time that is needed in order to guarantee that VM_i has executed at least t time units. T_{VM_i} is the period of VM_i, C_{VM_i} is the time VM_i has to execute each period, and R_{VM_i} denotes the worst-case response time of VM_i, i.e., the maximum time it may take until VM_i has executed C_{VM_i} time units after a release. Each VM_i, $(1 \leq i \leq k)$ runs a real-time program that consists of n_i tasks $\tau_{i,j}$ $(1 \leq j \leq n_i)$, i.e., $\tau_{i,j}$ denotes task j in VM_i. Each task $\tau_{i,j}$ is defined by its worst-case execution time $C_{i,j}$ and period $T_{i,j}$. For a fixed C_{VM_i}/T_{VM_i} our function and the method described here make it possible to find a maximal period T_{VM_i} for VM_i such that tasks meet their deadlines. Furthermore, we can also use our function and method for finding the minimum C_{VM_i}/T_{VM_i} given a T_{VM_i}, such that all tasks meet their deadlines.

We have included a detailed example that explains the reasons why our holistic method (called optimistic method in the paper) performs better than the previous approach that considered each VM in isolation (called pessimistic method in the paper). This quantifies how much the optimistic method saves resources than the pessimistic method.

If we do not consider context switching overhead, the minimum resource utilization is trivially obtained when the period of each VM is infinitely small. Having infinitely small VM periods is of course not realistic, and to be able to calculate the real optimal length of VM periods we have introduced an overhead model. Simulations show (and quantify) that the context switching overhead becomes more significant when many VMs share the same hardware resource.

Acknowledgements

This work is part of the research project "Scalable resource-efficient systems for big data analytics" funded by the Knowledge Foundation (grant: 20140032) in Sweden.



(a)

(b)

(c)

(d)

Figure 14(a): C_{VM}/T_{VM} versus Total utilization for 1 VM, (b) C_{VM}/T_{VM} versus Total utilization for 2 VMs, (c) C_{VM}/T_{VM} versus Total utilization for 4 VMs, (d) C_{VM}/T_{VM} versus Total utilization for 8 VMs

References

[1] L. Abeni and T. Cucinotta, “Efficient Virtualization of Real-Time Activities”, *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–4, 2011.

[2] E. Bini, T. H. C. Nguyen, P. Richard, and S. K. Baruah, “A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines”, *IEEE Transactions on Computers*, 58(2):279–286, Feb. 2009.

[3] E. Bini and G. C. Buttazzo, “Measuring the Performance of Schedulability Tests”, *Real-Time Systems*, 30(1–2):129–154, 2005.

[4] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*, Addison-Wesley Educational Publishers Inc, 2009.

[5] S. Chen, L. T. Phan, J. Lee, I. Lee, and O. Sokolsky, “Removing Abstraction Overhead in the Composition of Hierarchical Real-Time Systems”, *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 81–90, 2011.

[6] T. Cucinotta, G. Anastasi, and L. Abeni, “Respecting Temporal Constraints in Virtualised Services”, *33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC ’09.*, 2:73–78, 2009.

[7] T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi,

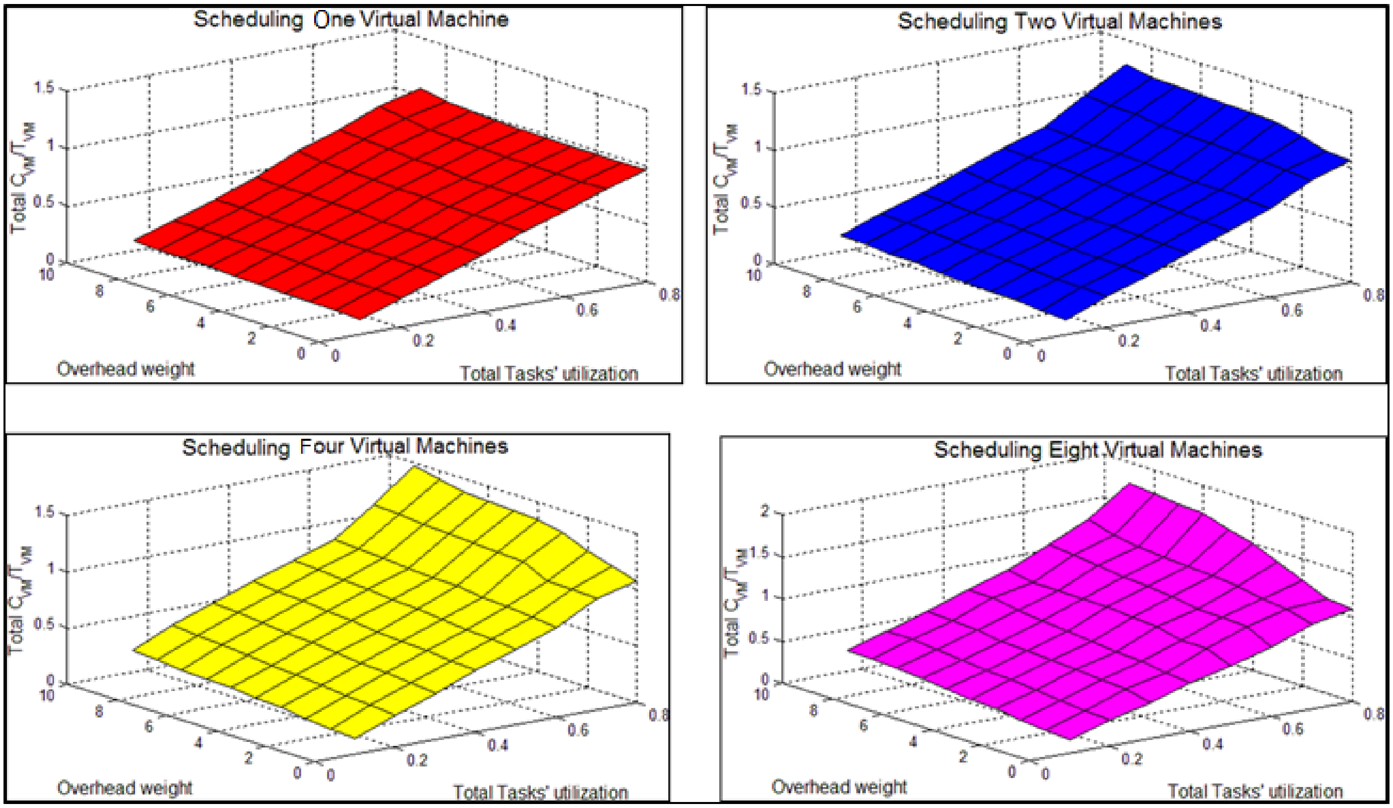


Figure 15: C_{VM}/T_{VM} versus Total utilization for the optimistic case

"Providing Performance Guarantees to Virtual Machines using Real-Time Scheduling", *Euro-Par 2010 Parallel Processing Workshops*, pp. 657–664, 2011.

[8] R. I. Davis and A. Burns, "Hierarchical Fixed Priority Pre-emptive Scheduling", *26th IEEE International Real-Time Systems Symposium, 2005. RTSS 2005*, 10 pp, 2005.

[9] K. J. Duda and D. R. Cheriton, "Borrowed-Virtual-Time (BVT) Scheduling: Supporting Latency-Sensitive Threads in a General-Purpose Scheduler", *ACM SIGOPS Operating Systems Review*, 33:261–276, 1999.

[10] X. Feng and A. K. Mok, "A Model of Hierarchical Real-Time Virtual Resources", *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pp. 26–35, 2002.

[11] M. Kutil, P. Sucha, R. Capek, and Z. Hanzalek, "Optimization and Scheduling Toolbox", *Matlab—Modelling, Programming and Simulations*, pp. 239–260, 2010.

[12] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting Soft Real-Time Tasks in the Xen Hypervisor", *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, New York, NY, USA, pp. 97–108, 2010.

[13] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing Compositional Scheduling through Virtualization", *2012 IEEE 18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 13–22, 2012.

[14] B. Lin and P. A. Dinda, "Vsched: Mixing Batch and Interactive Virtual Machines using Periodic Real-Time Scheduling", *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pp. 8, 2005.

[15] G. Lipari and E. Bini, "A Methodology for Designing Hierarchical Scheduling Systems", *Journal of Embedded Computing*, 1(2):257–269, 2005.

[16] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[17] S. Liu, G. Quan, and S. Ren, "On-Line Scheduling of Real-Time Services for Cloud Computing", *2010 6th World Congress on Services (SERVICES-1)*, pp. 459–464, 2010.

[18] L. Lundberg, "Analyzing Fixed-Priority Global Multiprocessor Scheduling", *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 2002*, pp. 145–153, 2002.

[19] L. Lundberg, "Utilization Based Schedulability Bounds for Age Constraint Process Sets in Real-Time Systems", *Real-Time Systems*, 23(3):273–295, 2002.

[20] L. Lundberg and S. Shirinbab, "Real-Time Scheduling in Cloud-Based Virtualized Software Systems", *Proceedings of the Second Nordic Symposium on Cloud Computing &*

Internet Technologies, pp. 54–58, 2013.

[21] W. Lunniss, S. Altmeyer, G. Lipari, and R. I. Davis, “Accounting for Cache Related Pre-Emption Delays in Hierarchical Scheduling”, *University of York, York, Technical Report YCS-2014-491 Available from http://www-users. cs. york. ac. uk/∼wlunniss*, 2014.

[22] J. Nieh and M. S. Lam, “A SMART Scheduler for Multimedia Applications”, *ACM Transactions on Computer Systems (TOCS)*, 21(2):117–163, 2003.

[23] L. T. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky, “Overhead-Aware Compositional Analysis of Real-Time Systems”, *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 237–246, 2013.

[24] H. Salimi, M. Najafzadeh, and M. Sharifi, “Advantages, Challenges and Optimization of Virtual Machine Scheduling in Cloud Computing Environments”, *International Journal of Computer Theory and Engineering*, 4(2),189-193, 2012.

[25] I. Shin and I. Lee, “Periodic Resource Model for Compositional Real-Time Guarantees”, *24th IEEE Real-Time Systems Symposium, 2003, RTSS 2003*, pp. 2–13, 2003.

[26] I. Shin and I. Lee, “Compositional Real-Time Scheduling Framework with Periodic Model”, *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):30, 2008.

[27] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, “A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems”, *17th IEEE Real-Time Systems Symposium, 1996*, pp. 288–299, 1996.

[28] P. Sucha, M. Kutil, M. Sojka, and Z. Hanzálek, “TORSCHE Scheduling Toolbox for Matlab”, *2006 IEEE Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006. IEEE International Symposium on Intelligent Control*, pp. 1181–1186, 2006.

[29] S. Xi, J. Wilson, C. Lu, and C. Gill, “RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen”, *2011 Proceedings of the International Conference on Embedded Software (EMSOFT)*, pp. 39–48, 2011.

[30] M. Xu, L. T. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. Gill, “Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms”, *2013 IEEE 34th Real-Time Systems Symposium (RTSS)*, pp. 1–10, 2013.

Christine Niyizamwiyitira is a PhD student in Computer Systems Engineering at Sweden’s Blekinge Institute of Technology (BTH) in the Computer Science and Engineering Department. Her research interests include: Real time systems, cloud computing, high performance computing, database performance, and voice based application. Her current research focuses on scheduling of real time systems on virtual machines (uniprocessor & multiprocessor) and big data processing.

Lars Lundberg is a Professor in Computer Systems Engineering at the Department of Computer Science and Engineering at Blekinge Institute of Technology in Sweden. He has a M.Sc. in Computer Science from Linköping University (1986) and a Ph.D. in Computer Engineering from Lund University (1993). His research interests include parallel and cluster computing, real-time systems and software engineering. Professor Lundberg's current work focuses on performance and availability aspects.

An Enhancement in Segmentation of Magnetic Resonance Images of Brain Tumors Using Symmetry and Active Contour

Mubbashar Saddique*
COMSATS Institute of Information Technology, Abbottabad, PAKISTAN

Kalim Qureshi†
Kuwait University, KUWAIT

Jawad Haider Kazmi*
COMSATS Institute of Information Technology, Abbottabad, PAKISTAN

Zainab Meraj†
Kuwait University, KUWAIT

Abstract

Brain tumor segmentation from Magnetic Resonance Imaging (MRI) is an important step toward surgical planning, treatment planning and monitoring of therapy. However, manual tumor segmentation commonly used in clinical practice is time consuming and challenging. Furthermore, none of the existing automated methods are highly robust, reliable, and efficient in a clinical setting. We present a segmentation technique named Algorithm using Symmetry Line and Active Contour (ASLAC). ASLAC has several advantages: (a) MRI image segmentation is performing automatically. (b) It exploits approximate left-right symmetry of the brain, (c) No preprocessing, such as intensity standardization or noise removal, is required by our algorithm, (d) It requires no labeled image data, nor any training, (e) It does not require image registration, (f) It can be implemented in real-time, (g) It can detect multiple tumors if tumor is fragmented into multiple parts of right/left side. Measured performance of ASLAC technique has improvement in terms of MRI image quality as compared to Chan-Vese [2] and HASA [19].

Key Words: MRI images, images segmentation, brain tumor segmentation, symmetry and active contour techniques, and image filters.

1 Introduction

Each year, more than 190,000 people in the United States and 10,000 people in Canada are diagnosed with a brain tumor. Of these, over 40,000 are primary brain tumors. In the United States, the overall incidence of all primary brain tumors is 14.1 per 100,000 people per year. Primary brain tumors are the leading cause of solid tumor cancer deaths in children under the age of 20. The National Brain Tumor Foundation (NBTF) for research in the United States estimates the death of 13,000 patients while 29,000 undergo primary brain tumor diagnosis every year [20]. The detection of brain and tumor tissue in Magnetic Resonance (MR) images and Computed Tomography (CT) scan images has been an active research area. Brain image segmentation from MRI images is problematical and challenging, but its precise segmentation is necessary for tumor detection and classification [14]. MRI's are the most efficient imaging technique used for early detection of abnormalities in the brain.

Today MRI brain tumor detection has significant importance. Doctors and radiologists can miss the abnormality due to inexperience in the field of cancer or tumor detection. The pre-processing is the most important step in MRI brain image analysis due to poor captured image quality. Pre-processing is necessary to correct and adjust the image for further study and processing. Different types of filtering techniques are available for pre-processing. These filters are normally used to improve the image quality, suppress the noise, preserves the edges in an image, enhance and smoothen the image. For this purpose, median, adaptive median, average or mean, and wiener filter used for MRI brain image pre-processing.

Brain tumor segmentation consists of separating the different tumor tissues (solid or active tumor, edema, and necrosis) from normal brain tissues: gray matter (GM), white matter (WM), and cerebrospinal fluid (CSF). In brain tumor studies, the existence of abnormal tissues may be easily detectable most of the time. However, accurate and reproducible segmentation and characterization of abnormalities are not straightforward. In the past, many researchers have made significant survey in the field of medical imaging, soft computing and brain tumor

---

* Department of Computer Science. Email: mubashar.chaudary@gmail.com, jawadkazmi@ciit.net.pk.

† Department of Information Science, College of Computing Sciences and Engineering. Email: kalimuddinqureshi@gmail.com, z.almeraj@gmail.com.

ISCA Copyright© 2017

segmentation [1, 12, 21]. Both semiautomatic and fully automatic methods have been surveyed in [12]. Clinical acceptance of segmentation techniques has depended on the simplicity of the segmentation, and the degree of user supervision. Interactive or semiautomatic methods are likely to remain dominant in practice for some time, especially in these applications where erroneous interpretations are unacceptable. We have been studying MRI/MRA imaging [10, 16, 18] and the work presented in this paper is an extension of our paper published in [19].

A healthy brain has a strong sagittal symmetry that is weakened by the presence of tumor. The comparison between the healthy and ill hemisphere, considering that tumors are generally not symmetrically placed in both hemispheres, was used to detect the anomaly. Supervised, unsupervised and registration based techniques required prior knowledge and user interaction for segmentation [19] but we have proposed a solution using symmetry and active contour that produce better results than above discussed techniques which does not require prior knowledge and user interaction.

The rest of the paper is organized as follows, in Section 2, we briefly describe classification of brain tumor image segmentation techniques. In Section 3, we describe the related work, in Section 4, we presented HASA and ASLAC proposed scheme. The measured results are outlined in Section 5 and discussion on results is described in Section 5. Section 6 concludes the paper.

2 Brain Tumor Image Segmentation Methods Classification

Brain tumor segmentation methods can be classified into three categories according to the degree of required human interaction as described [5] manual segmentation, semiautomatic segmentation, and fully automatic segmentation. In the section below we describe the advantages and limitations of category.

2.1 Manual Segmentation

In manual segmentation, human experts (radiologists/anatomists/trained technologists) not only make use of the information presented in the image but also make use of additional knowledge such as anatomy. Manual delineation requires software tools with sophisticated graphical user interfaces to facilitate drawing regions of interest and image display. In practice, the selection of the tumor region, which is the region of interest (ROI), is a tedious and time-consuming task. If the person drawing the ROI is not a radiologist/anatomist/trained technologist who is well versed with that brain anatomy, it will most likely yield poor segmentation results. The task of marking the tumor regions slice by slice sometimes limits the human rater’s view and generates jaggy images. As a result, the segmented images are less than optimal showing a “stripping" effect [15].

2.2 Semiautomatic Segmentation

In semiautomatic brain tumor segmentation, the intervention of a human operator is often needed to initialize the method, to check the accuracy of the result, or even to manually correct the segmentation result. Most of the current research is targeted at semiautomatic segmentation of brain tumors with the intention of having the least human interaction possible. According to [5], the main components of an interactive brain tumor segmentation method are the computational part, the interactive part, and the user interface. The computational part corresponds to one or more pieces of program capable of generating a delineation of the tumor given some parameters. The interactive part is responsible for mediating information between the user and the computational part. It translates the outcome produced by the computational part into visual feedback to the user and the data input by the user into parameters for the program.

2.3 Fully Automatic Segmentation

In fully automatic methods, the computer determines the segmentation of tumor without any human interaction. Fully automatic methods generally incorporate human intelligence and prior knowledge in the algorithms, and are usually developed making use of soft computing and model-based techniques such as deformable models. The study of automatic brain tumor segmentation represents [7, 22] an interesting research issue in Machine Learning and Pattern Recognition, since it represents a problem that humans can learn to do effectively. However, developing highly accurate automatic methods remains a challenging problem. For automatic segmentation, it is essential to have a model that not only describes the size, shape, location and appearance of the tumor but that also permits expected variations in these characteristics. However, no completely automatic segmentation algorithm has yet been adopted in the clinic environment.

3 Review of Current Generation Tumor Image Segmentation Techniques

The segmentation of brain tumor images is a challenging task for several reasons. Firstly, high-grade gliomas usually exhibit unclear and irregular boundaries with discontinuities. Secondly, after contrast injection, contrast uptake and image acquisition time can vary, which changes tumor appearance significantly however, the non-imaginable component of the tumor should be handled by segmentation algorithms. In recent years a great effort of the research in the field of medical imaging was focused on brain tumor segmentation.

Image 4 segmentation refers to the process of partitioning an image into groups of pixels that are homogeneous with respect to some criterion. The result of segmentation is the splitting up of the image into connected areas. Thus, segment is concerned with dividing an image into meaningful regions. Here in this section, we will describe the efforts of other researchers in tumor segmentation techniques such as detection of brain tumor by wavelet based image fusion, fuzzy symmetry based genetic clustering technique, wavelet and FCM algorithm, random walk, modified kernelized fuzzy c-mean, multiple kernel fuzzy c-means, neural network used for image segmentation, and symmetric technique.

3.1 Detection of Brain Tumor based on Wavelet Based Image Fusion

The objective of image fusion is to combine information from multiple images of the same scene. The result of image fusion is a new image which is more suitable for human and machine perception or further image-processing tasks such as segmentation, feature extraction and object recognition. Different fusion methods have been proposed in literature, including multi-resolution analysis. In [7], wavelets based image fusion algorithm was applied to the MR images and CT images, used as primary sources to extract the redundant and later information to enhance the brain tumor detection in the resulting fused image.

In [6], an image fusion technique was effectively used to detect the tumor in a mixture of complex backgrounds. Image fusions were applied by merging multiple images resulting into precise information about the size, shape and placement of the tumor.

3.2 Fuzzy Symmetry Based Genetic Clustering Technique for MRI Brain Image Segmentation

In [11], an automatic segmentation technique of multispectral image of the brain is proposed using a fuzzy symmetry based on genetic clustering. A developed fuzzy point symmetry based cluster validity index, fuzzy symmetry index, was used to measure ‘goodness’ of the corresponding partition. The genetic fuzzy clustering technique evolves the number of clusters present in the data set automatically. The proposed method was better when compared with Fuzzy C-means. This method does not require any priority specification of the number of clusters present in the data set. The obtained results are compared with the available ground truth information. This method segments the membership values of points to different clusters which are calculated by the point symmetry based distance rather than the Euclidean distance. This method is used to automatically develop the proper clustering of all types of clusters, both convex and non-convex, which have some symmetrical structures.

3.3 Wavelet and FCM Algorithm for MRI Brain Image Segmentation

In [4, 9], Fuzzy C-Means (FCM) Clustering and wavelet decomposition for the feature extraction and feature vector are treated as input to FCM. This method is called Wavelet Fuzzy Fuzzy C-Means (WFCM) and is used to segment the tumor from MRI. WFCM involves two stages, one is feature extraction and the other is clustering. The feature extraction was processed by using multilevel 2D wavelet decomposition features. Extracted features from wavelet decomposition are forwarded to FCM for the purpose of clustering.

3.4 Random Walk MRI Brain Image Segmentation

Random walk is defined as discrete random motion in which a particle repeatedly moves a fixed distance up, down, left and right. This is a region growing based image segmentation method based on random walk of a particle. In this method, the initial position at which a particle is initially present is known as seed point movement from one position to another method based on the probability calculations. This method has three pragmatic properties that are, weak boundary detection, noise robustness and the assignment of ambiguous regions. Seed point selection is very important for random walk, after the seed point has been detected random walk methods performed better segmentation [13].

3.5 Modified Kernelized Fuzzy C-Means (MKFCM) Segmentation for MRI Images

Image topology and statistical parameters of a window around the pixel are also considered to modify the algorithm. MKFCM was directly applied to image segmentation like FCM. The MKFCM algorithm was implemented for MRI image segmentation [8].

3.6 Multiple Kernel Fuzzy C-Means (MKFC)

The fuzzy c-means is a popular soft clustering method. Applying kernel behavior, the kernel fuzzy c-means algorithm attempts to address the problem by mapping data with nonlinear relationships to correct feature spaces. Kernel combination, or choice, is critical for efficient kernel clustering. Unfortunately, for most applications, it is not easy to find the right combination [3].

3.7 Neural Network Based Methods

Artificial Neural Networks (ANNs) were developed for a variety of applications such as function approximation, feature extraction, optimization, pattern recognition and classification. Mostly, they have been developed for image enhancement, segmentation, registration, feature extraction, and object recognition. From the above applications, image segmentation is more important and a crucial step for high level image processing such as detection of tumors in medical images. Hopfield, Back Propagation Networks (BPN), Self-Organized Map (SOM), Multi-Layer Perceptron (MLP), Radial Basis Function (RBF), Adaptive Resonance Theory (ART), Cellular, and Pulse-Coupled neural networks have been used for image segmentation [17].

3.8 Symmetry Imaging

Symmetry is an important characteristic to identify the structure of the object and has been used in many domains. In the medical imaging field symmetry was mainly applied to the brain imaging where a healthy brain has symmetry in the right and the left side of a brain image. Symmetry imaging has many advantages over other imaging techniques for example; this technique has not required preprocessing, labeled image data, training and registration as other techniques are required. Symmetry based brain tumor investigation was carried out in [19].

4 Hybrid Symmetry Approaches [19]

The proposed technique is an extension of our research paper that was published in [19]. A flowchart of HASA (Hybrid Algorithm using Symmetric and Active contour) is shown in Figure 1 and in complete detail [19] for the readers understanding.

4.1 Proposed Algorithm Using Symmetry Line and Active Contour (ASLAC)

In ASLAC we cut the image into two parts left and right along the symmetry line. Compute the reflection image of left or right part opposite of tumor side i.e., (if tumor is present on right side take reflection of left side and if tumor is present on left side then take reflection of right side). In the next step we find the difference of the tumor side and reflection image. We apply the threshold value and get the binary image on the newly created image. We map the image on the original left or right side of the image. After that we apply an active contour and find the boundary of the abnormal region. The proposed ASLAC technique flow chart (shown in Figure 2) and its detailed steps are shown in Section 4.1. The Chan-Vese [2], HASA [19] and proposed ASLAC techniques component comparison is listed in Table 1.

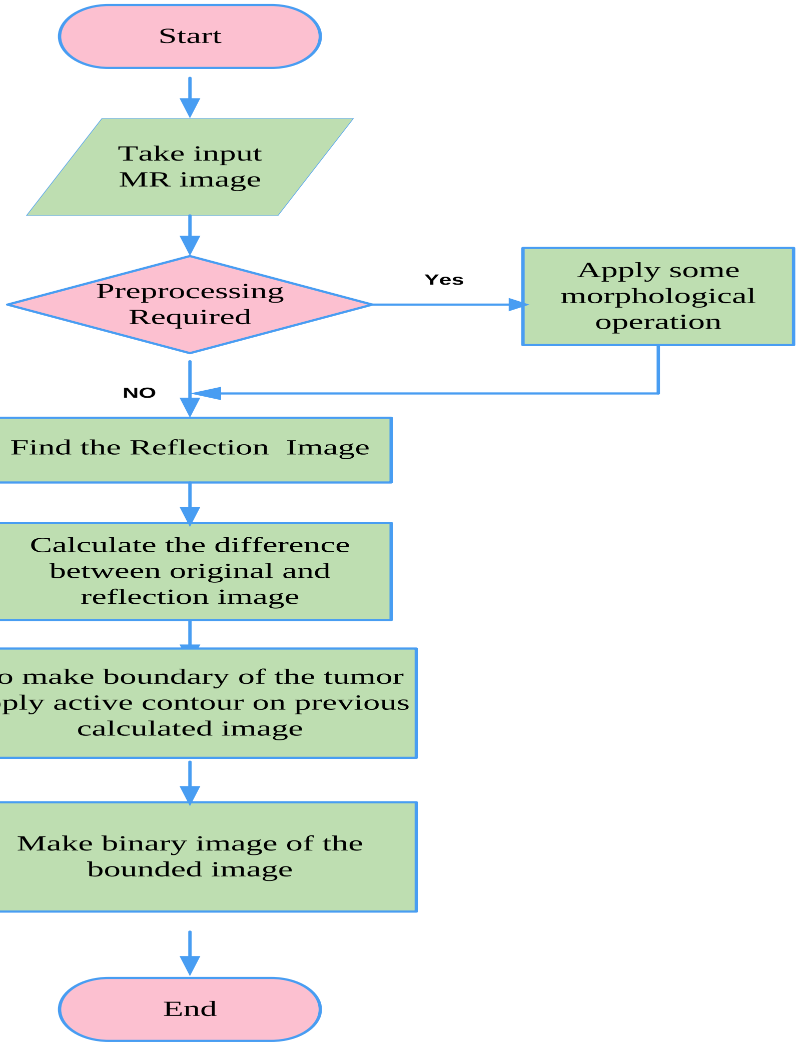

Start
Take input MR image
Preprocessing Required
Yes
Apply some morphological operation
NO
Find the Reflection Image
Calculate the difference between original and reflection image
To make boundary of the tumor apply active contour on previous calculated image
Make binary image of the bounded image
End

Figure 1: Flow chart of HASA

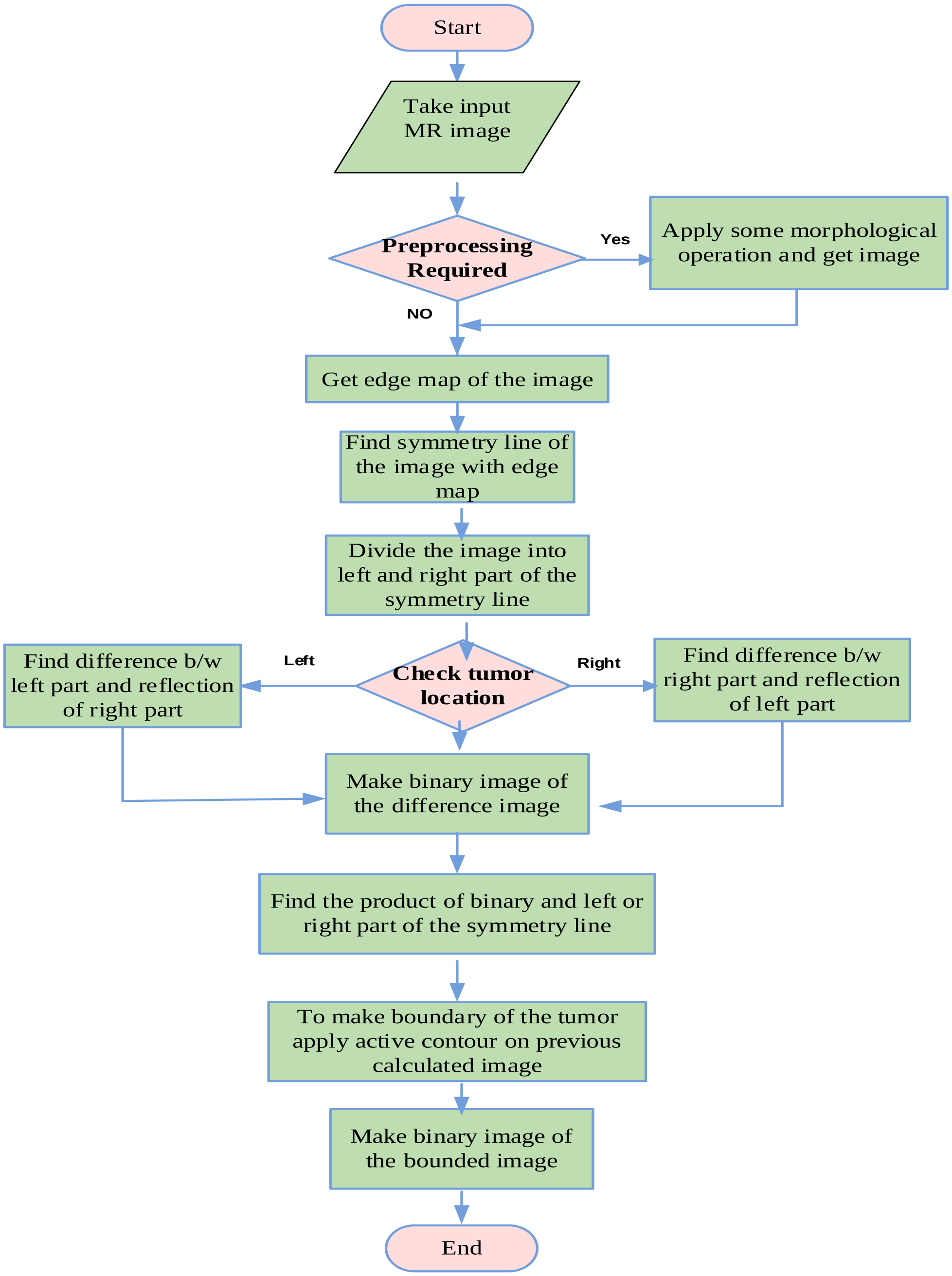

Start
Take input MR image
Preprocessing Required
Yes
Apply some morphological operation and get image
NO
Get edge map of the image
Find symmetry line of the image with edge map
Divide the image into left and right part of the symmetry line
Left
Check tumor location
Right
Find difference b/w left part and reflection of right part
Find difference b/w right part and reflection of left part
Make binary image of the difference image
Find the product of binary and left or right part of the symmetry line
To make boundary of the tumor apply active contour on previous calculated image
Make binary image of the bounded image
End

Figure 2: Flow chart of ASLAC Algorithm

Table 1: Chan-Vese[2], HASA[19] and proposed ASLAC techniques main component based comparison

Chan-Vese	**HASA**	**Proposed ASLAC**
1. Apply some morphological operations to preprocess the image. 2. Apply the active contour and segment the tumor	1. Apply active contour on difference of images between actual and reflection of actual image.	1. Find Symmetry line of image from edge map. 2. Divide the image into left and right part of the symmetry line. 3. Find tumor location in left or right part of image. 4. Find the difference between right/left with reflection image of right/left part. 5. Make binary image of difference. 6. Apply active contour on only left or right part on which side tumor exist.

```
1. Apply morphological operations, like erosion and dilation, if preprocessing needed.  After this optional step let our image be denoted by O(x; y).
2. Find the edges of the image.
3. Find the symmetry line with the following ways
    • Find location of 1 from each row in the matrix of edge map image
    • Now calculate the average location of each row
    • Now from these avg values find the line by the formula
        $Y = \alpha + \beta x$ ---------- (i)
Where
        $\alpha = \frac{\sum_{i-1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i y_i \sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^{2}}$
        $\beta = \frac{n \sum_{i-1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^{2}}$
4. Separate the right and left part along the symmetry line
    • For Left side
        i.  Start outer loop 1 to maxcol or maxrow
        ii. Start inner loop 1 to symmetryline
            • Put all the values in new matrix of location outer and inner loop
    • For Right side
        i.  Start outer loop 1 to maxcol or maxrow
        ii. Start inner loop symmetryline+1 to end of maxcol or maxrow
            • Put all the values in new matrix of location outer and inner loop
5. Find the reflection image of both right and left part i.e Right(x,y) and Left(x,y)  which are calculated in step 4
    • If tumor is right side then
  New image= Rightimage – Reflection of  leftimage
    • Else
  New image1= Leftimage – Reflection of  rightimage
6. Find the  threshold with the formula
   Avg= max { O(x,y) }*0.25
7. Now find the mask of  newimage(x,y) or newimage1(x,y) using the threshold value
        If ( N (x,y) or N1(x,y)>avg)
            N (x,y) or N1(x,y)=1
        Else
            N (x,y) or N1(x,y)=0
8. Map the mask with original image O(x,y) i.e multiply the mask and original image.
9. Now find the location of the tumor
    • Find max intensity in the new image
    • Find the locations of max intensity, and apply active contour
```

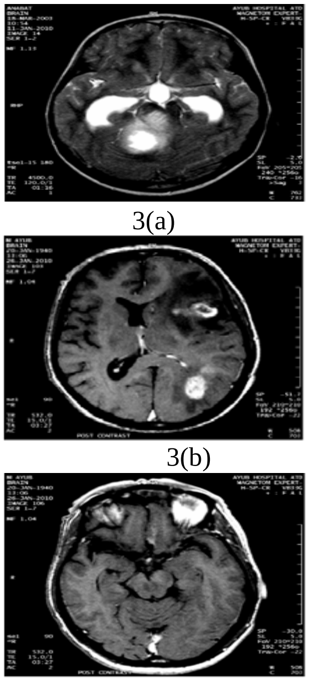
Pseudocode of proposed ASLAC Technique

5 Results

We have applied the ASLAC technique on DICOM formatted MRI data of 3 different patients. The specifications of all three data sets were the same and are listed in Table 2. The investigated original images are shown in Figures 3(a), 3(b), & 3(c). The images produced by applying the ASLAC technique is shown in Figure 4. Figure 4(a) is the original DICOM image which after morphological pre-processing yields the image in Figure 4(b). Figure 4(c) is the edge map of Figure 4(b) after applying canny algorithm. Figure 4(d) yields the image after applying the symmetry line. Figures 4(e) and 4(f) is the right and left part of Figure 4(d). Figure 4(g) is the reflection part of Figure 4(f). Figure 4(h) is the difference of

Table 2: Parameter values of MRI images dataset

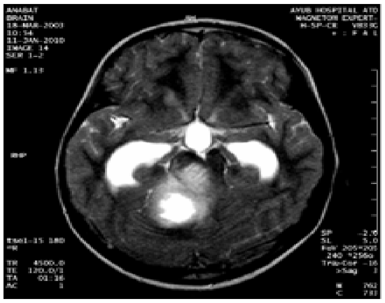
Parameters	**Dataset 1-3 Values**
Magnetic Field Strength	1.5
File Size	1049472
Format	DICOM
Width	1024
Height	1024
Bit Depth	8
Color Type	grayscale
Modality	'MR'
Samples Per Pixel	1
Photometric Interpretation	Monochrome2
Rows:	1024
Columns	1024
Pixel Aspect Ratio	[2x1 double]
Bits Allocated	8
Bits Stored	8
High Bit	7
Pixel Representation	0
Window Center	127.5000
Window Width	255

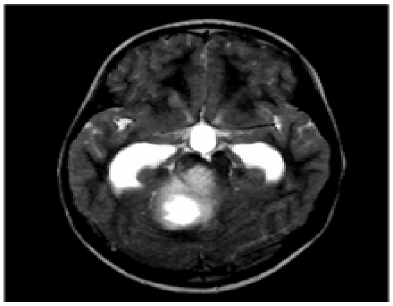

3(a)

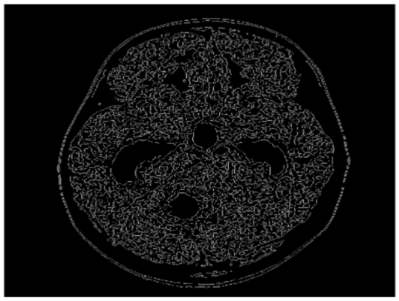
3(b)

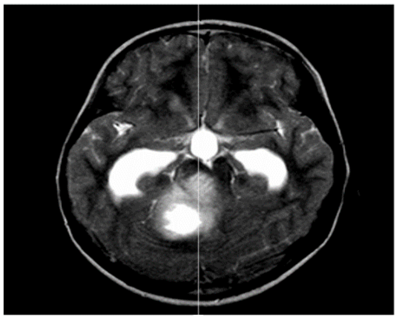
3(c)

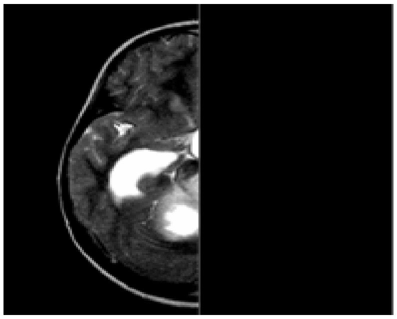
Figure 3: 3(a), 3(b), and 3(c) are original images

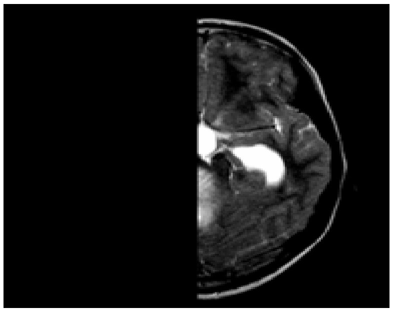

4(a): Original slice of size 1024x1024 before morphological operation

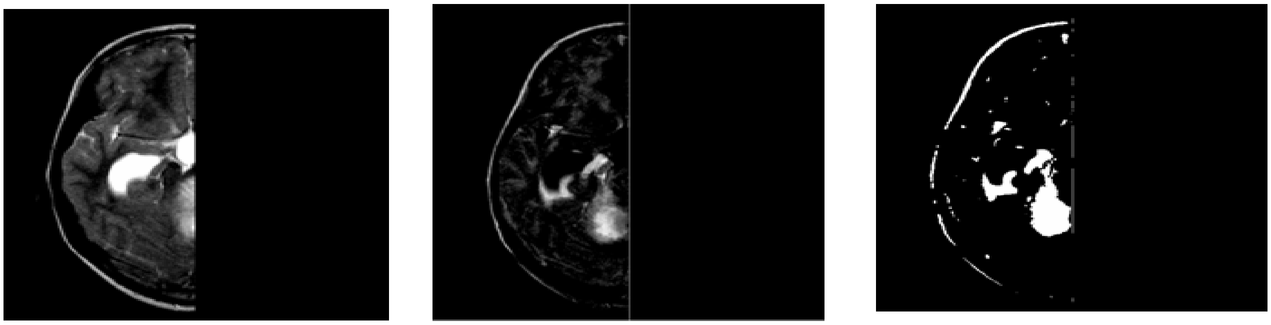

4(b): After morphological operations


4(c): Edge map of Figure 4b


4(d): Image with symmetry line


4(e): Right part of Figure 4(d)

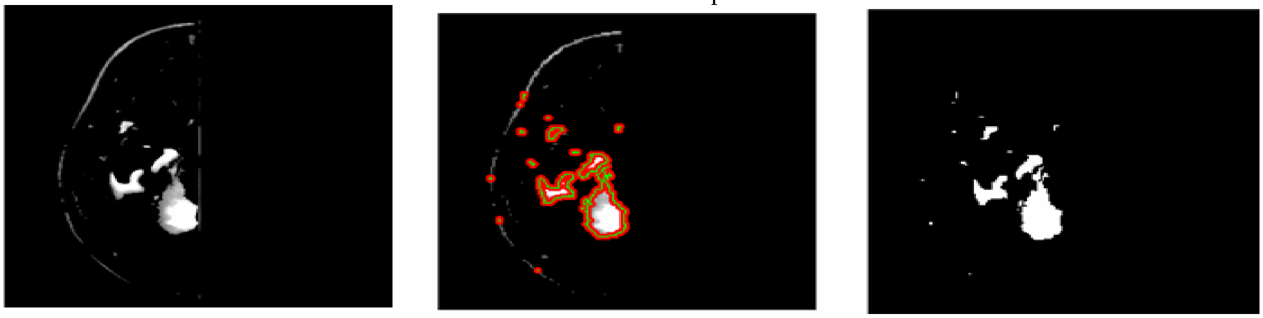

4(f): Left part of Figure 4(d)



4(g): Reflection image of figure 4f

4(h): Difference between right part and reflection of left part

4(i): Binary image of figure 4(h)



4(j): Mapping of figure 4i and 4e

4(k): Boundary of the tumor after applying the active contour

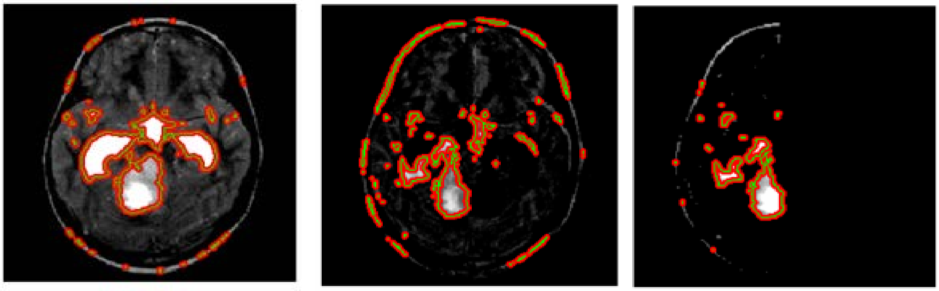
4(l): Binary image of figure 4(k)

Figures 4(a)-4(l): Represents the step by step images produced by applying ASLAC technique

Figures 4(e) and 4(g). Figure 4(i) is the binary image of Figure 4(h) by applying the threshold formula. Figure 4(j) is the product of Figures 4(i) and 4(e). Figure 4(k) yields an image after applying the active contour [2]. Figure 4(l) depicts the binary image of Figure 4(k).

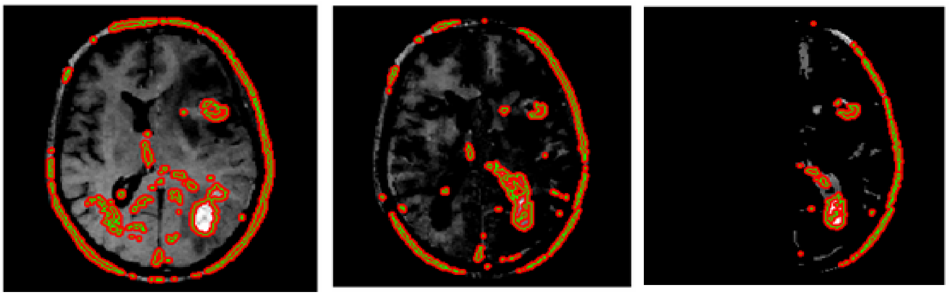
6 Discussion

By applying Chan Vese algorithm [2] on images 3(a), 3(b) & 3(c) the results produced are listed in Figures 5(a), 6(a), and 7(a), and by applying HASA [19] the produced images are shown in Figures 5(b), 6(b), 7(b). The segmentation quality is improved as compared to the images produce by applying Chan-Vese algorithm. The results produced by applying ASLAC technique is shown in Figures 5(c), 6(c), and 7(c). By examining the segmentation quality produced by applying Chan-Vese (Figures 5(a), 6(a), 7(a)), HASA (Figures 5(b), 6(b), 7(b)), and ASLAC (Figures 5(c), 6(c), 7(c)), the images produced by applying the ASLAC technique has shown improvement. Each image is improved in terms of filtering unwanted information and the contrast of tumor region is also improved as seen in Figures 5(c), 6(c) and 7(c).



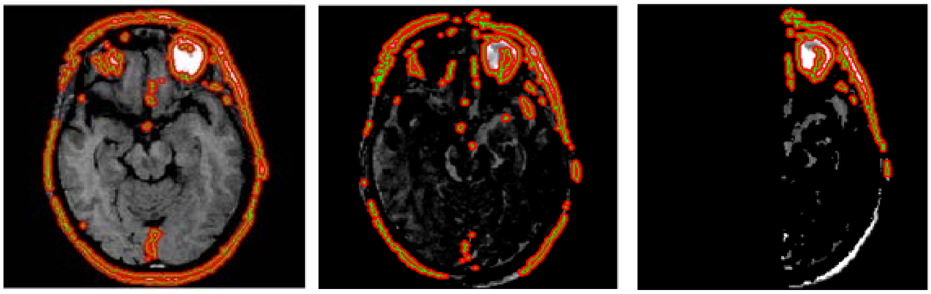
(a) Chan-Vese (b) HASA (c) ASLAC

Figure 5: Images produced by applying Chan-Vese, HASA, EHASA, ASLAC on image data set 1



(a) Chan-Vese (b) HASA (c) ASLAC

Figure 6: Images produced by applying Chan-Vese, HASA, EHASA, and ASLAC on image data set 2



(a) Chan-Vese (b) HASA (c) ASLAC

Figure 7: Images produced by applying Chan-Vese, HASA and ASLAC on image data set 3

The images produced by using the ASLAC algorithm are better as compare to the Chan-Vese, and HASA, techniques. In ASLAC technique, the image is divided into parts and applied to the active contour only that part which has a tumor, either the left or the right. So in this way we have a better segmentation of the brain tumor and the number of iterations are also reduced by half as compared to the Chan-Vese, and HASA techniques.

7 Conclusion

The proposed ASLAC technique can identify the tumor/abnormality in either the right or the left side and can also find more than one tumor. This technique does not require any user interaction and is fully automatic. The images produced by ASLAC have a better result as compared to other well-known existing techniques. One limitation of our technique is that it will not produce true results if the tumor is present on the symmetry line. In the future, we will address the deficiencies of the ASLAC technique.

References

[1] Stefan Bauer, Roland Wiest, Lutz-P Nolte, and Mauricio Reyes, “A Survey of MRI Medical Image Analysis for Brain Tumor Studies”, *Physics in Medicine and Biology*, 58:97-129, 2013.

[2] T. Chan and L. Vese, “Active Contours without Edges”, *IEEE Transactions on Image Processing*, 10(2):266-277, 2001.

[3] L. P. Clarke, R. P. Velthuizen, M. A Camacho, J. J. Heine, and M. Vaidyanathan, “MRI Segmentation: Methods and Application”, *Magnetic Resonance Imaging*, 13:343-368, 1995.

[4] Moumen T. El-Melegy and Hashim M. Mokhtar, “Tumor Segmentation in Brain MRI using a Fuzzy Approach with Class Center Priors”, *EURASIP Journal of Image and Video Processing*, 2014:21, 2014.

[5] Nelly Gordillo, Eduard Montseny, and Pilar Sobrevilla, “State of the Art Survey on MRI Brain Tumor Segmentation”, *Magnetic Resonance Imaging*, 31:1426-1438, 2013.

[6] Leo Grady, Random Walk for Image Segmentation”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28:1768-1783, 2006.

[7] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle, "Brain Tumor Segmentation with Deep Neural Networks”, Medical image analysis, 35:18-31, 2017.

[8] Hsin-Chien Huang, Yung-Yu Chuang and Chu-Song Chan, “Multiple Kernel Fuzzy Clustering”, IEEE Transactions on Fuzzy Systems, 2011.

[9] I. Irakykhalifa, Aliaa Youssif, Howida Youssry, “MRI

Brain Image Segmentation based on Wavelet and FCM Algorithm", *International Journal of Computer Applications*, 47(16):32-39, 2012.

[10] J. H. Kazmi, K. Qureshi, and H. Rasheed, "An Implementation of SAN Filter and Edge Sharpening Method for MRA Images", *Malaysian Journal of Computer Science*, 20:99-114, 2007.

[11] Chunming Li, Rui Huang, Zhaohua Ding, J. Chris Gatenby, and Dimitris N. Metaxas, "A Level Set Method for Image Segmentation in the Presence of Intensity in Homogeneities with Application to MRI", IEEE Transactions on Image Processing, 20:2007-2016, 2011.

[12] J. Liu, M. Li, J. Wang, F. Wu, T. Liu, and Y. Pan, "A Survey of MRI-Based Brain Tumor Segmentation Methods", Tsinghua Science and Technology, 19:578-595, 2014.

[13] Abdenour Mekhmoukh, Karim Mokrani, and Mohamed Cheriet, "A Modified Kernelized Fuzzy C-Means Algorithm for Noisy Images Segmentation: Application to MRI Images", *IJCSI International Journal of Computer Science Issues*, 9(1):1694-0814, 2012

[14] J. J. Pedroso De Lima, "New Trends in Medical Imaging", Radiation Protection Dosimetry", 115(1-4):51-57, 2005.

[15] M. Prastawa, E. Bullitt, N. Moon, K. Van Leemput, and G. Gerig, "Automatic Brain Tumor Segmentation by Subject Specific Modification of Atlas Priors", AcadRadiol, 10(12):1341–1348, 2003.

[16] Sonia Rauf, "A Study of Vessels Extraction Techniques from MRA", MS Thesis, supervised by Jawad H. Kazmi and Kalim Qureshi, Comsats Institute of Information Technology, Abbottabad, Pakistan, 2013.

[17] N. Ray, B. N. Saha, and M. R. Graham Brown, "Locating Brain Tumors from MR Imagery Using Symmetry Signals", Systems and Computers, pp. 224–228. 2007.

[18] Mubbashar Saddique, "Classification and Segmentation of Brain Abnormality from MRI Using Symmetry", MS Thesis, supervised by Jawad H. Kazmi and Kalim Qureshi, Comsats Institute of Information Technology, Abbottabad, Pakistan, 2012.

[19] Mubbashar Saddique, Jawad Haider Kazmi, and Kalim Qureshi, "A Hybrid Approach of Using Symmetry Technique for Brain Tumor Segmentation", *Journal of Computational Mathematical Methods in Medicine*, Article ID 712783, 2014, http://dx.doi.org/10.1155/2014/712783.

[20] Amarjot Singh, Srikrishna Karanam, Shivesh Bajpai, Akash Choubey, and Thaluru Raviteja, "Malignant Brain Tumor Detection", 4th IEEE International Conference on Computer Science and Information Technology, *IEEE Xplore*, 1:163-167, 2011.

[21] S. Valarmathy, R. Ramani, and N. Suthanthira Vanitha, "A Survey of Recent Image Segmentation Techniques for MRI Brain Images", *IJCST*, 4(1):2013.

[22] Lian Yanyun and Song Zhijian, "Automated Brain Tumor Segmentation in Magnetic Resonace Imaging based on Sliding-Window Technique and Symmetry Analysis", *Chinese Medical Journal*, 127:462-468, 2014.

Mubbashar Saddique is working as Deputy Director (IT) at Punjab Employees Social Security Institution, Lahore, Punjab, Pakistan. He completed a B.Sc (Telecommunication Engineering) from Institute of Engineering & Technology, Lahore Campus, Pakistan. He received a merit scholarship from COMSATS Institute of Information Technology, Abbottabad, Pakistan where he completed his MS computer science in 2012. His area of research interest is imaging processing, networks and data mining. His contact email is mubashar.chaudary@gmail.com.

Kalim Qureshi is an Associate Professor of Information Science Department, Kuwait University, Kuwait. His research interests include network parallel distributed computing, thread programming, concurrent algorithms designing, task scheduling, performance measurement and medical imaging. Dr. Qureshi receive his Ph.D and MS degrees from Muroran Institute of Technology, Hokkaido, Japan in (2000, 1997). He published more than 40 journal papers in reputed journals. His email addresses: kalimuddinqureshi@gmail.com, and kalim_qureshi@hotmail.com.

Jawad Haider Kazmi is an Assistant Professor at the Department of Computer Science, CIIT Abbottabad, Pakistan since 2008. He received a MIT in 2003 and MS in Computer Science in 2007. His research interest includes medical imaging, distributed systems and computer networks. His contact email is jawad.kazmi@yahoo.com.

Zainab Meraj (Photo not available). She is working in Kuwait University, Information Science Department. She is an active researcher in area of computer graphics.

Using an Uncalibrated Camera for Undistorted Projection over a Mobile Region of Interest

Mamona Awan*
National University of Computer and Emerging Sciences – FAST,
Lahore, 54700 PAKISTAN

Kwang Hee Ko†
Gwangju Institute of Science and Technology
Gwangju, 61005 REPUBLIC OF KOREA

Abstract

This paper presents a system for projection over a mobile region of interest. The region is supposed to be located on an arbitrarily placed planar surface. The system consists of a projector, a laptop and an uncalibrated camera. Our method is based on homography concepts and computes mappings between a camera, a projector and the region. The technique excludes calibration parameters of the camera and the projector; hence, the intrinsic and extrinsic parameters are neither given nor calculated. The positions of the projector, the camera, and the surface are unknown. For the numerous unknowns, we show that each mapping corrected distortion can be achieved. The system is capable of dynamically identifying the shape of the region of interest in which projection is on, while the region moves with a certain constant speed. Upon a sudden and abrupt motion of the region of interest, the projection iteratively adjusts its position until it fits the region perfectly. The system is demonstrated with real examples.

Key Words: Undistorted projection, uncalibrated camera, mobile surface projection, projector-camera system.

1 Introduction

Projector systems are widely used for the purpose of entertainment, work, and even as public displays. They often retain a few problems; one of the problems is a distorted projection. Unless the projector is carefully aligned with respect to the display area, the projected image appears distorted. The second problem is a fixed projection; the projector is unable to project over a mobile surface unless a special device is developed that tracks the position and orientation of the moving surface. Such problems may limit the application of a projector system. In order to solve this problem, a device like a camera can be considered to obtain the information of the position and the orientation of the surface. In this way, we can make the projection system efficient enough such that it can form an undistorted projection and project the undistorted images over a mobile surface.

There are several systems for correcting the distortion or the keystone problem of a projector as in [12], [14] and [15]. A number of systems are also designed for the integration of multiple images projected by multiple projectors to form a large seamless image as in [2], [3] and [11]. There is another research dedicated for the pose estimation of the projector using images obtained by a camera using the screen-camera homography [9]. Although these systems can provide efficient methods for projecting undistorted images on a surface, they do not handle dynamic projection. The dynamic projection includes the projection over randomly moving objects or region of interests. Another system is designed using light sensors and a frequency variation of the projector light, using a complex pre-calibration process, which needs hefty calculations and special apparatuses [8].

There exists another method for calibrating a projector using structured light patterns over the screen embedded with light sensors. This approach has two major drawbacks; first, the use of light sensors requires other communication modules to transmit necessary data to the computer; secondly the system needs to recalibrate as the position of the projection surface changes. This recalibration process is done by projecting a series of structured light patterns again [7]. Other studies have explored tracking techniques of a moving surface for projection of content; however, they primarily depend upon either electromagnetic sensors or a number of visual based tracking systems that add a significant amount of cost, infrastructure and complexity to achieve such an effect as done in [1] and [13]. Optical and magnetic trackers are used in [1] for the tracking of the projectable object while in [13] they have used photo-sensing wireless tags such as active radio-frequency (RF) tags.

In this paper, we propose a system for projection over a mobile region of interest exclusive of such complexities. Our system consists of a laptop connected to a projector and an uncalibrated camera independent of any costly sensors and trackers for the purpose of mobile projection. The mobile region of interest is supposed to be moving within a planar

---

* E-mail: mamonaawan@nu.edu.pk.
† E-mail: khko@gist.ac.kr.

domain, and through an image processing method the mobile region is detected and then used as a display region for projection. The image fits the moving region of interest with proper alignment, exclusive of any distortion or bleeding. The system is also capable of correcting the keystone distortion, given only a display screen is in consideration without any mobile region of interest. Our system excludes hefty calculations of calibrations of the camera, as well as the projector. The intrinsic and extrinsic elements of the projector and the camera are totally ignored and the process excludes all of such relations with these elements.

2 Overall Procedure

The process flow for projection over an arbitrarily placed planar surface is shown in Figure 1. The process starts by taking an image of planar surface without any projection over it via camera; the planar surface may or may not include the region of interest (ROI) at this point. This image is processed to identify the boundary of the planar surface. As the aspect ratio of the length and width of this planar surface is known, we can form a homography relation between the camera and the planar surface. We refer to this homography as H_{cs} in the paper.

The second process (process 2) in Figure 1 includes projecting a source image to determine the transformation between the projector domain and the camera domain. The position of the planar surface is assumed to be such that when the source image is projected, a certain region of the source image is visible over the planar surface. An image of this projection is taken by the camera. The source image and the camera image are then used to formulate a homography relation between the projector and the camera, which is denoted as H_{pc}. Once these two homographies, i.e., H_{cs} and H_{pc} are calculated, we can form a projection over this planar surface without any distortion.

The homography relations can be summarized as:

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} \approx H_{pc} \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \quad (1)$$

and

$$\begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \approx H_{cs} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix} \quad (2)$$

In (1) and (2), (x_p, y_p), (x_c, y_c), and (x_s, y_s) are the corresponding points in the projector domain, the camera domain, and the planar surface domain, respectively. The homography between two domains $H_{D_1D_2}$ (where "D_1" and "D_2" represent two random domains), is an invertible 3x3 matrix. Hence, a relation between the projector and the planar surface can be formed using (1) and (2). The effect of H_{cs} can be excluded from the relation calculated previously, as the relation linking the projector and the camera H_{pc} is calculated using the planar surface amid. This gives us the relation between the projector and the surface as a 3x3 matrix, mathematically expressed as:

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} \approx {H_{cs}}^{-1} * H_{pc} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix} \quad (3)$$

In Equation (3) ${H_{cs}}^{-1}$ represents the inverse of the homography H_{cs}.

The system then continues and determines if any region of interest is present within the planar surface or not. As shown in Figure 2, if the region of interest is absent, then the system considers the same surface as a display screen and merely

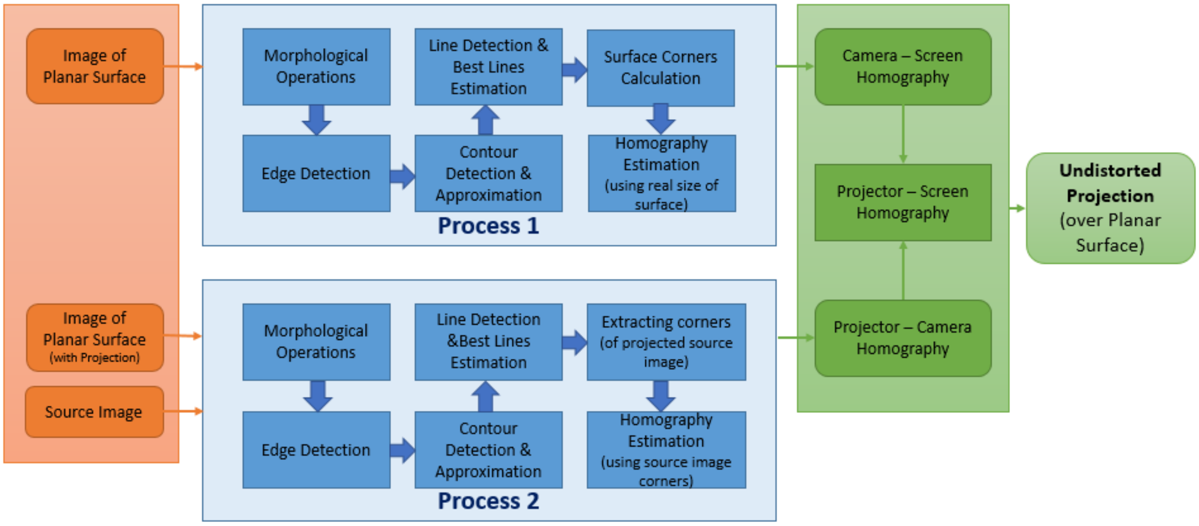


Image of Planar Surface
Image of Planar Surface (with Projection)
Source Image
Morphological Operations
Line Detection & Best Lines Estimation
Surface Corners Calculation
Edge Detection
Contour Detection & Approximation
Homography Estimation (using real size of surface)
Process 1
Morphological Operations
Line Detection &Best Lines Estimation
Extracting corners (of projected source image)
Edge Detection
Contour Detection & Approximation
Homography Estimation (using source image corners)
Process 2
Camera – Screen Homography
Projector – Screen Homography
Projector – Camera Homography
Undistorted Projection (over Planar Surface)

Figure 1: Process flow for projection over arbitrarily placed planar surface

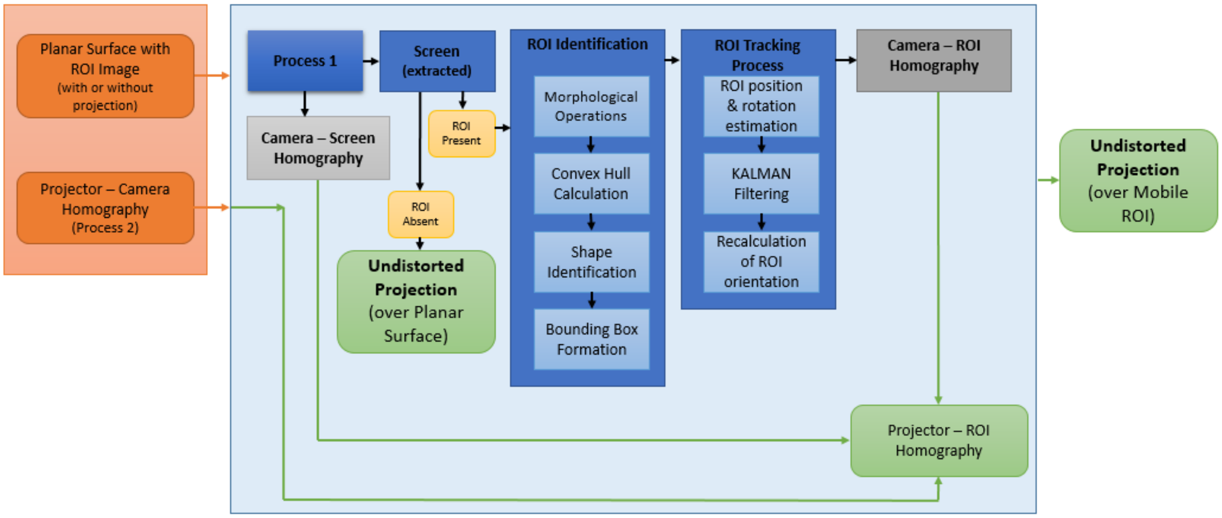

Planar Surface with ROI Image (with or without projection)
Projector – Camera Homography (Process 2)
Process 1
Screen (extracted)
Camera – Screen Homography
ROI Present
ROI Absent
Undistorted Projection (over Planar Surface)
ROI Identification
Morphological Operations
Convex Hull Calculation
Shape Identification
Bounding Box Formation
ROI Tracking Process
ROI position & rotation estimation
KALMAN Filtering
Recalculation of ROI orientation
Camera – ROI Homography
Projector – ROI Homography
Undistorted Projection (over Mobile ROI)

Figure 2: Process flow for projection over mobile ROI in planar surface

projects over it. If the planar surface is moved, the display modifies with respect to it as well, up to a certain extent.

If the region of interest is present, then the system follows the process flow presented further in Figure 2. It determines the position and orientation of region of interest in the surface domain. The process begins with application of morphological operations over the image that has already planar surface area extracted. Then we extract contours and form a convex hull over the contours detected. Using these contours, we identify the shape of the ROI and also form a bounding box to estimate and refine its position and orientation in the planar surface. As the original aspect ratio of the ROI is known, so we can form a homography H_{sroi} for the ROI and planar surface. The expression can be stated as:

$$\begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix} \approx H_{sroi} \begin{pmatrix} x_{roi} \\ y_{roi} \\ 1 \end{pmatrix} \tag{4}$$

Where (x_s,y_s) and (x_{roi},y_{roi}) are the corresponding points in the surface domain and the ROI domain respectively. H_{sroi} can be used to determine the exact orientation of the ROI on the surface. The relation between the projection domain and the ROI can also be formed by using this homography. For this computation, we modify the previously calculated mapping in (3), so the mapping or homography between the ROI and a projector can be stated as:

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} \approx H_{sroi} * {H_{cs}}^{-1} * H_{pc} \begin{pmatrix} x_{sroi} \\ y_{sroi} \\ 1 \end{pmatrix} \tag{5}$$

$$H_{proi} = H_{sroi} * {H_{cs}}^{-1} * H_{pc} \tag{6}$$

Where H_{proi} is the homography relation between the projector and the ROI. This homography can then be used to pre-warp the image to be projected such that it fits the region of interest without any distortion or bleeding. However, homography will only give us the true mapping for the ROI, so the scale and the correct position should also be considered in order to fit the projection correctly without any bleeding.

3 Technical Details

In order to project over the mobile surface, a few assumptions are made for our system. The mobile surface is supposed to be any of the three shapes (rectangular, triangular, or circular). It should have a white surface with black boundary to simplify identification. The system is not configured for any other shape but a few modifications can enable it to recognize other shapes as well. The shape is unknown by the system as a priori but has to be any of the above mentioned. The size of this surface is arbitrary but the aspect ratio of the length and width is assumed to be known. Only one region of interest can be introduced over the planar surface at a time. The mobile region of interest (ROI) is supposed to be moving within a planar surface. For these experiments we have taken a white board (non-reflective type) with boundaries marked with black, as a planar surface. This board can also be used as a conventional projector screen; however, its dimensions are significantly smaller as compared to a conventional projector screen. The size of screen is irrelevant and can be random; however, the aspect ratio of the size (length: width) is assumed to be known.

The tiresome task of calibrating the camera by taking pictures of a checkerboard is completely excluded; hence for any instance we do not need any intrinsic or extrinsic parameters of the camera. The relative position of the projector and the camera does not change once the system has started; however, for another instance, the relative position of the camera and the projector may vary. The experiments are performed in a projection cubicle that is a dark room with the projector being the only light source. The application is valid

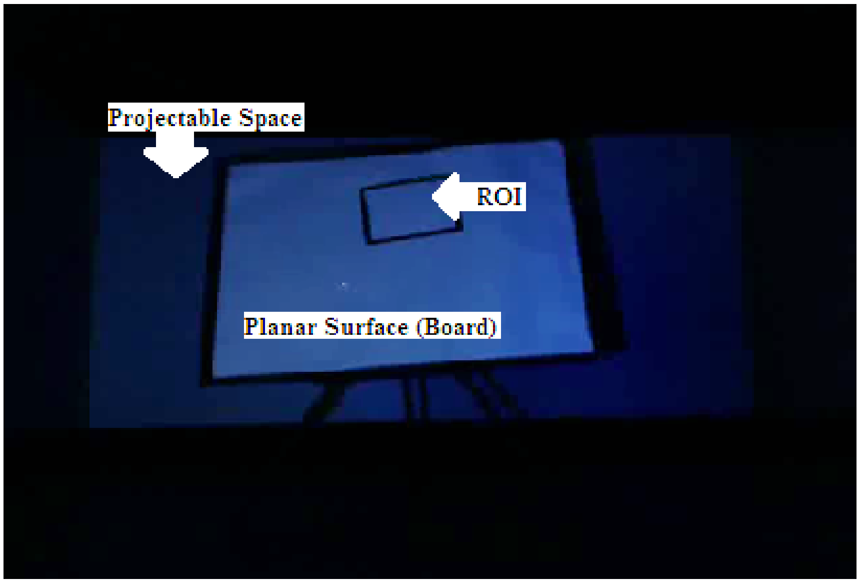


Figure 3: Planar Surface (Board) and the Region of Interest (ROI) within it, as seen by the camera. Projector (not shown) is projecting a black image to represent projectable space. The visibility of this image is improved for the sake of illustration

for most of the instances with minor changes in threshold values used in the program which can be set manually.

3.1 Homography Calculation

While a projection is viewed via a camera, any point in the projector domain can be mapped to a certain point in the camera domain. This mapping from the projected image to the captured image or from the captured image to the projected image, form a homograph H, a 3x3 matrix. As for any point in a domain we can calculate its corresponding point in the other domain as follows:

$$(x, y) = \left(\frac{H_{11}x' + H_{12}y' + H_{13}}{H_{31}x' + H_{32}y' + H_{33}}, \frac{H_{21}x' + H_{22}y' + H_{23}}{H_{31}x' + H_{32}y' + H_{33}}\right) \quad (7)$$

Where (x, y) is a point in a domain and (x', y') is its corresponding point in the other domain. The parameters H_{11},…,H_{33} are the unknowns to be determined as follows:

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \quad (8)$$

Although there are 9 unknown parameters, there are only 8 DOF (degree of freedom) as $\sum_{ij} H_{ij} = 1$. Hence, only four corresponding pairs of points are required, as each point provides two constraints. If four pairs of corresponding points are attained, then we can obtain a unique solution for these parameters by using a direct linear transformation algorithm (DLT).

3.1.1 Camera-Planar Surface Homography. In order to calculate camera-planar surface homography, the camera captures an image of the planar surface. This image is used to detect the boundary of the planar surface and the corners are extracted. The aspect ratio of the surface is already known. We compute the maximum length of the surface in the camera domain and calculate four corresponding points using the known aspect ratio. These corners are then used to calculate the homography “H_{cs}”, between the camera and the surface.

We can convert each point from the surface domain to the camera domain and vice-versa using the expression stated previously in (2). If only a planar surface is provided (a planar surface does not have ROI within it), then the system uses the surface as a display screen and forms an undistorted projection over it. If the surface is moved up to some extent during this projection, the system can modify the projection simultaneously to project over it correctly; however, the main focus of the system is to project over ROI.

3.1.2 Projector-Camera Homography. Many techniques are followed to form a relation between a projector and a camera. These techniques usually include projecting a series of images via the projector and capturing these by the camera. The images may contain structured light patterns, AR (augmented reality) tags or even calibration grids. We exclude the cumbersome task of projecting a series of images as it takes a lot of time for a user, to project at least 10 images in the case of structured light and 3 images in the case of AR tags. We have also excluded the projection of calibration grids as it incorporates other correspondence techniques as well.

Our method includes projection of a simple image that can be reproduced without any complex considerations. The image is supposed to have a white rectangle with a black background as shown in Figure 4. It should also have dimensions similar to the resolution of the projector, so that the projector does not apply any transformation on it before projecting it. Each pixel in the image is supposed to represent each pixel in the projector domain.

The system is capable of camera exposure adjustment and the user is also given a provision to adjust the exposure, if necessary. An image of projection is taken by the camera and morphological operations are applied over it to extract the edges and register contours. These contours are then approximated into another simpler contour using the Ramer-Douglas-Puecker algorithm [10]. This approximation significantly improves the detection of lines in the contour due to reduced curves in the contour.

The lines are detected by applying Hough Lines Transform over the approximated contour and then the best fitting lines are selected using a minimum least square error computation [4]. The lines are used for the calculation of four corners of the white rectangle. These corners are then sorted in the clockwise manner and correspond with the points of the source image. The simplicity of the source image provides such convenience that we can find corresponding points in the source image by merely using a corner extraction algorithm and a sorting algorithm.

These four-point correspondences are enough for the calculation of homography H_{pc}.

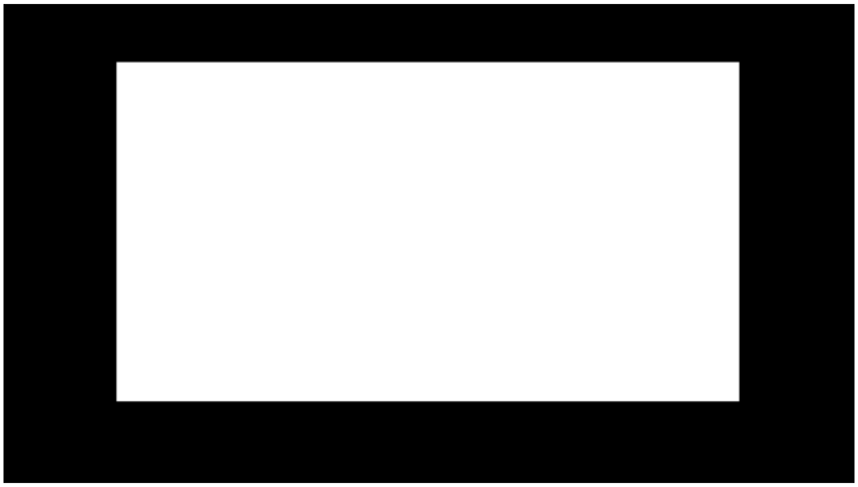


Figure 4: The source image used for projector-camera homography calculation. The image dimensions are 1920 by 1080 pixel

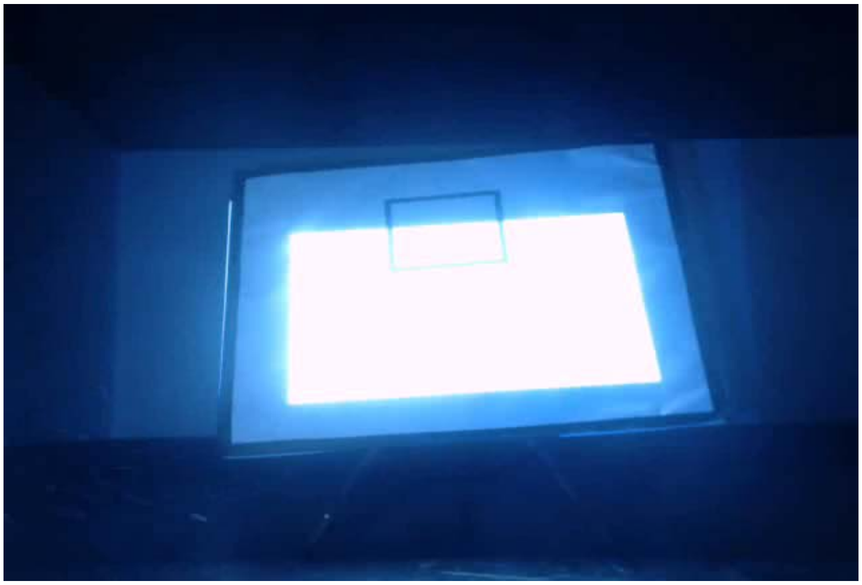


Figure 5: Projection of source image as viewed by the camera. The region of interest can also be seen in this image. The presence of ROI does not affect the calculation of projector-camera homography

Each point in the camera image or domain can be converted into the corresponding point in the projector domain using (1).

3.2 Initial Pose Estimation

Now that the system can already map from a projector to a camera and the camera to a planar surface, the only computation left is to form a mapping between the planar surface and the region of interest within it. For this purpose, the system analyzes the same image captured for camera-planar surface homography calculation. By applying different morphological operations, the boundary of the ROI is stored as a contour and approximated into a much simpler contour by the Ramer-Douglas-Peucker algorithm. This approximation makes it easy for us to form a better convex hull with the least computation time, as the number of vertices is significantly reduced by the approximation. This reduction in the number of vertices can be seen in Figure 6. The approximated contour also eliminates the complexity of smaller curves in the convex hull to be formed.

The convex hull is then used to identify the shape of the ROI. The vertices of the convex hull and the interior angles at each vertex are calculated along with the calculation of Hu Moments [5] of the convex hull. The number of vertices, interior angles at vertices, and Hu moments of the convex hull are the three parameters that decide the shape of the ROI. The shape of ROI can either be a rectangle, a triangle or a circle. Once the shape of ROI is decided, we form a bounding box over the convex hull of ROI. For the case of circular ROI, we form an upright bounding box (un-rotatable), as a rotatable bounding box will be redundant in this case. We form a rotatable bounding box for other shapes of ROI.

The introduction of the bounding box solves a number of problems for us. As the convex hull changes with each frame, we apply Kalman filter of order 2, over the center and the rotation angle of our corresponding bounding box [6].

The filter avoids all the abrupt changes in the position or the rotation of the bounding box and provides a smooth variation in both of these values. Furthermore, the size of convex hull also changes as each frame is processed. This variation in size is caused by different miscellaneous effects like intensity variation caused by the simultaneous projection. So, we calculate the size of the corresponding bounding box as the average of the previously determined sizes along with the recent size. Figures 6 and 7 are excellent examples to demonstrate the importance of forming a bounding box over the convex hull. Although, the convex hull in Figure 6 lacks one corner completely, the bounding box still manages to provide the fourth corner as seen in Figure 7. The corners of this bounding box can then be used to form homography H_{sroi} as explained earlier in (4).

3.3 Image Rendering for Projection

Using the calculated mappings, we can form a pre-warp for the image to be projected; however, simply pre-warping the image will not be enough. The process of rendering the image starts by scaling the image to be projected, as homography does not give us the true scale of the image. The true scale of the image depends on the resolution of the projector, as each pixel in the image should light only one pixel in the projector domain. The system then warps the image by the inverse of H_{proi} calculated in (6). The dimensions of the image to be projected are kept the same as the resolution of the projector. The image is rendered by placing the transformed image at such a position Pos_{roi}, that it coincides with the ROI. The rest

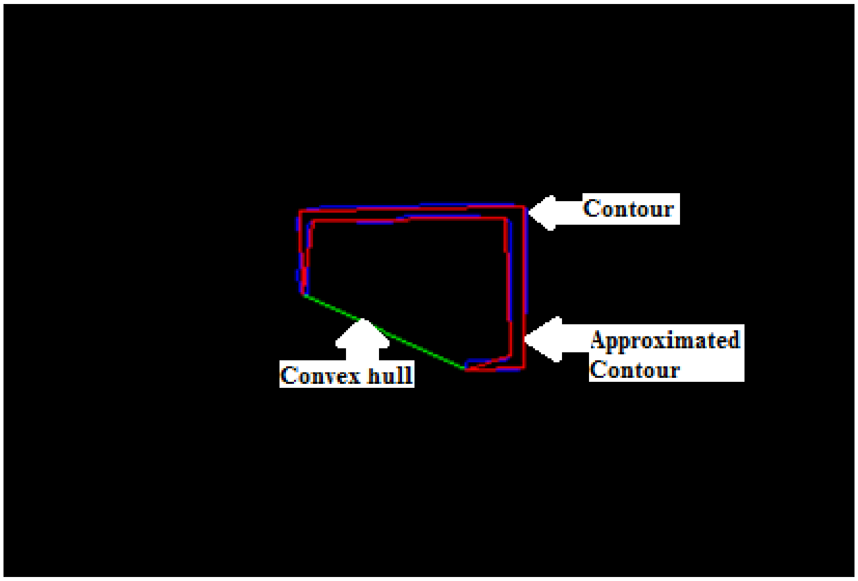


Figure 6: ROI (rectangular) is being detected in planar-surface. Contour of ROI (Blue), contour after approximation by Ramer-Douglas-Peucker algorithm (Red), and convex hull of the approximated contour (Green) can be seen in the image. The image is taken as seen in the planar surface domain

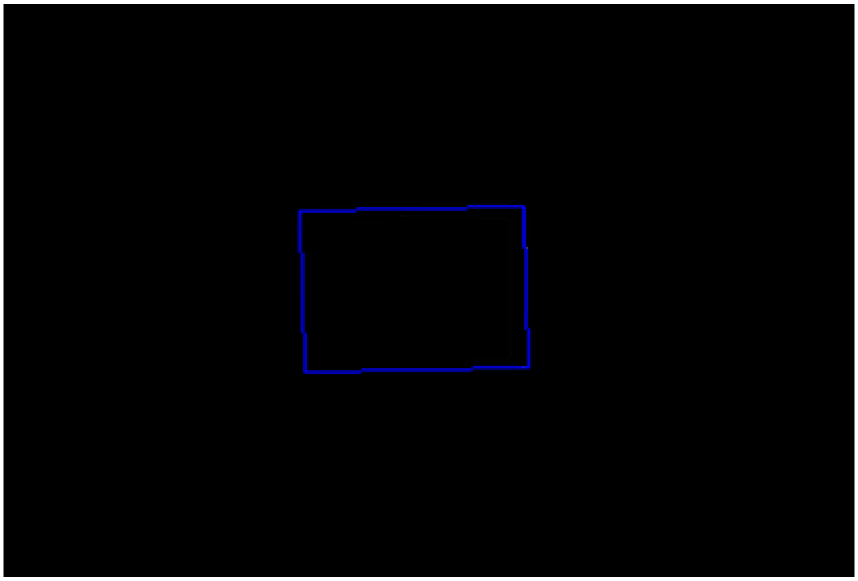


Figure 7: The bounding box for ROI (rectangular). The corresponding contours and convex hull can be seen in Figure 6. The image is shown as seen in planar-surface domain

of the pixels are rendered as black.

To calculate Pos_{roi}, we take in the position of ROI in planar surfaces domain and convert it into the projector domain using the inverse of H_{proi}. This will give us the position Pos_{roi}, as stated in (9).

$$Pos_{roi} = {H_{proi}}^{-1} * Pos_{sroi} \tag{9}$$

In (9), Pos_{sroi} is the position of ROI as seen in the planar surface domain and ${H_{proi}}^{-1}$ represents the inverse of the mapping transformation calculated in (6). Figure 8 shows an example of the rendered image for projection.

4 Implementation

The projector used in our implementation is EPSON LCD Projector EB-1776W, Model H476C. The projector provides a projection at resolution of 1920 by 1080 pixels at 50-60 Hz of refresh rate. The brightness of the projector is 3000 lumens for both white and color pixels. The camera we have used is Logitech HD Pro Webcam C920 operating at 30 fps (frames per second). The webcam images are captured with the resolution of 640 by 480 pixels with a bit depth of 24. The camera also has an automatic low-light correction feature; however, this feature is disabled for these experimentations.

The application is written in C++ using OpenCV Library and implemented using a Lenovo ThinkPad E440, with Microsoft Windows 8 Pro. It has an Intel core i5-4210M processor working at 2.60GHz. The laptop has also NVidia GeForce 840M installed in it; however, GPU modules have not been used for this application. The planar surface and the region of interest both are made of a foam board, covered with a paper sheet of A0 and A4 sizes respectively.

The foam boards are considered because of their light weight and easy mobility. The boundaries of these boards are painted black. The dimensions of the foam boards for the planar surface and ROI are A0 and A4 respectively. The experiments are performed in a cubicle with only a projector as the light source. The camera is placed near the projector and the board is mounted over a poster stand. The board has an arbitrary inclination away from the setup. The distance of the board from the setup is taken to be arbitrary and has changed between different experimental instances as well.

4.1 Reprojection Error

As the homography we have calculated is just an approximation of the relation between the two domains, hence its projection image rendered using this approximated homography has a reprojection error. In Figure 9, the green circle represents the top-left corner of the ROI in the camera co-ordinates and the red circle represents the top-left corner of the ROI in planar surface co-ordinates. It can be vividly seen that the projected image does not coincide properly over the ROI. To solve this issue, we calculate the reprojection error in terms of Euclidean distance and then threshold it under an acceptable range. The rendered image was then modified by eliminating this error and the image overlaid over the ROI properly.

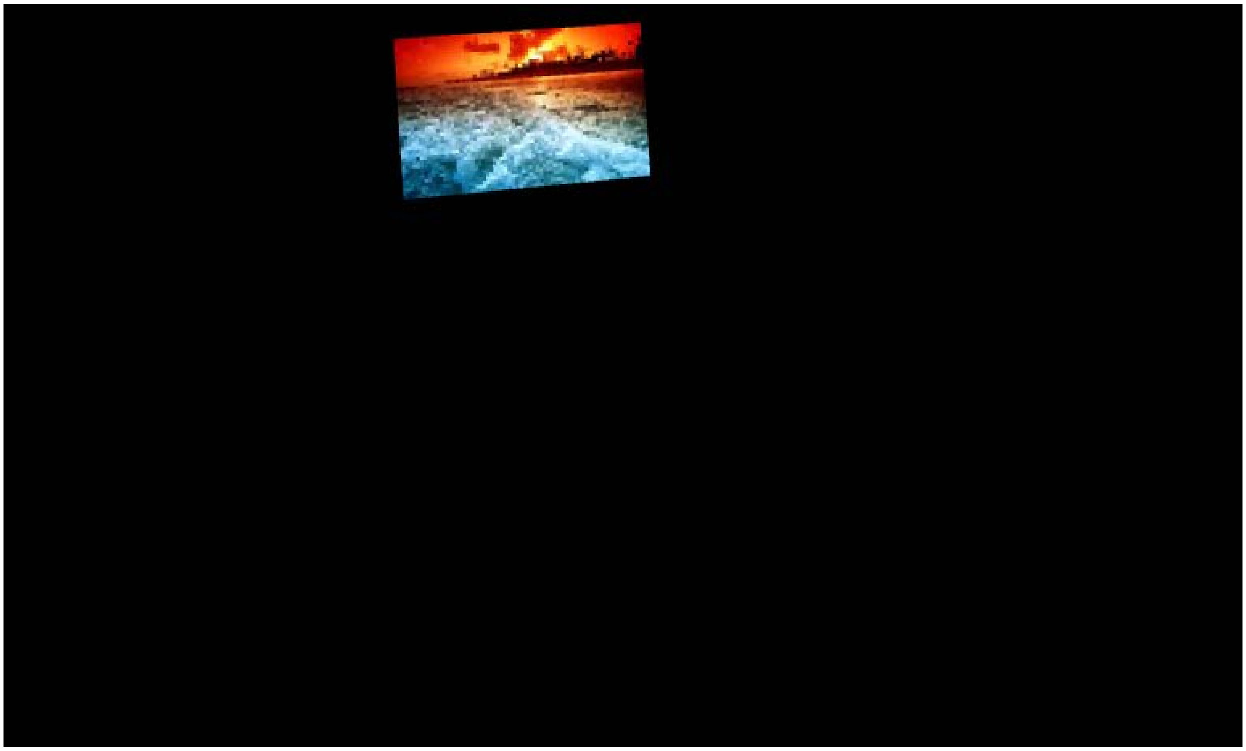


Figure 8: Sample of image rendered for projection over ROI (rectangular). The rendered image has a resolution of 1920 x1080 pixels (same as our projector)

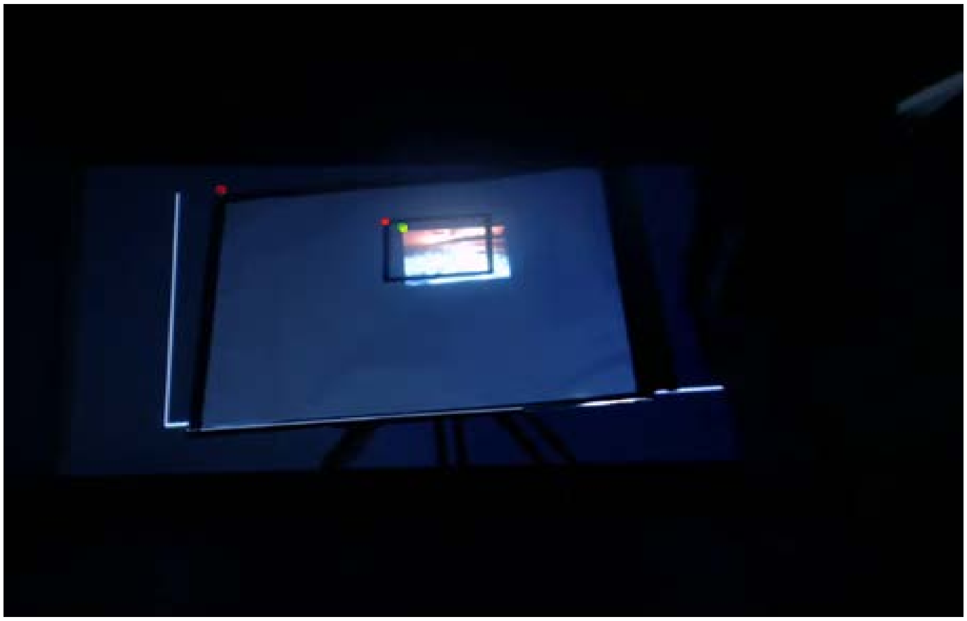


Figure 9: The projected image does not coincide with the actual boundary

4.2 Image Clipping Issue

To render the image, we have used inverse transformation; however, if ROI is rotated at a larger angle then inverse transformation can cause the image to rotate out of the visible range of the of the image plane. This can result in image clipping as shown in Figure 10 where it rotates out of the ROI at the top-right corner and in Figure 11 where it rotates out at the bottom-left corner. To cater to this problem, we calculate the inverse transformation for the boundary pixels first and

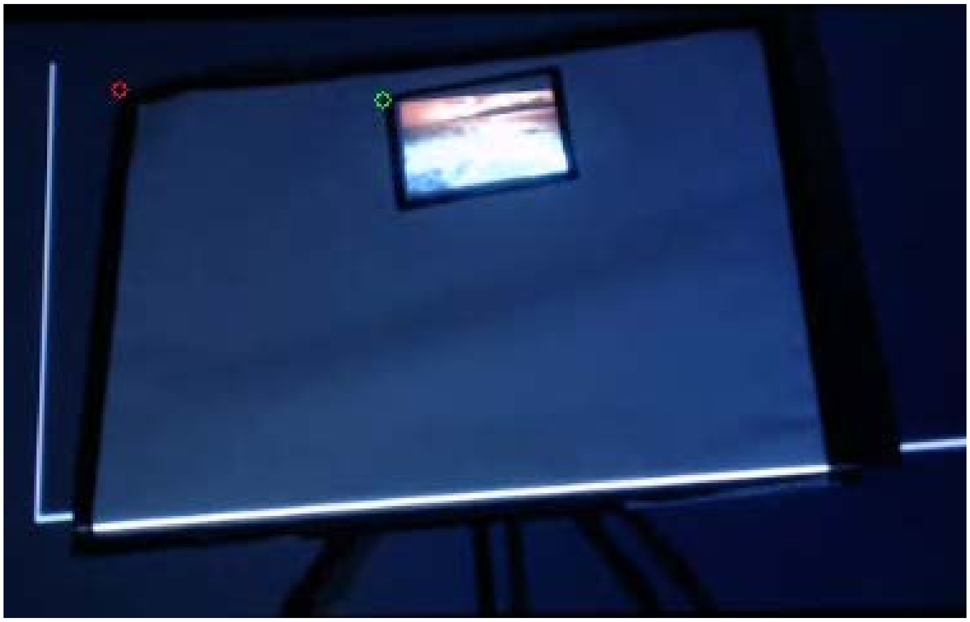


Figure 10: Image clipped in top-right corner of ROI

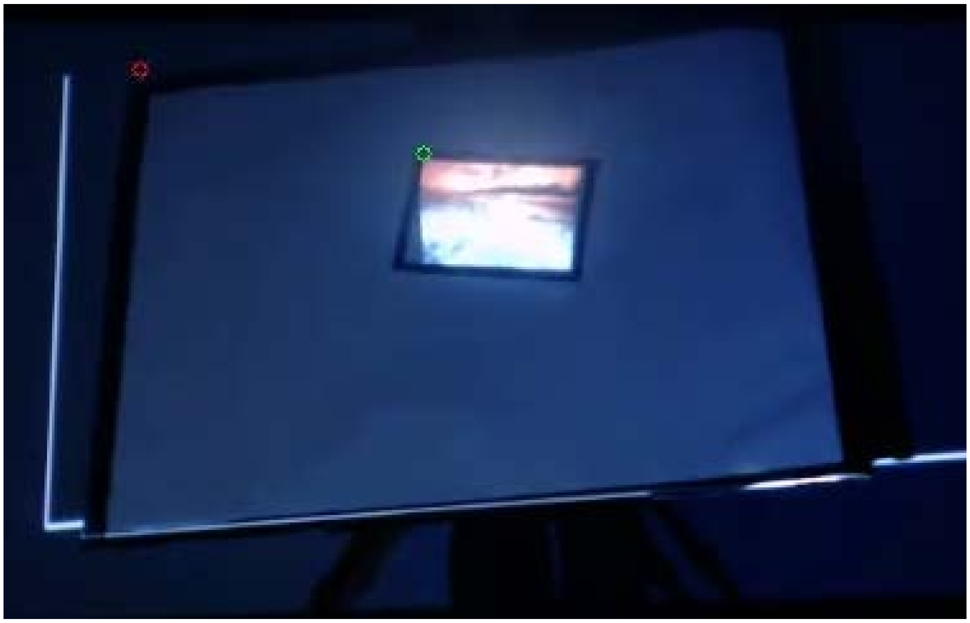


Figure 11: Image clipped in bottom-left corner of ROI

check if those pixels are in visible range. If those pixels transform out of the visible range, we calculate the difference (d_x, d_y) and modify the translation of our transformation matrix. Whenever the ROI is oriented in a rotated manner, this modification allows us to refrain the clipping of the projected image.

5 Results

The system is tested for several instances and it takes less than 13 seconds to project and capture images. The projection converges efficiently; however, the use of a Kalman filter has significant effect over the pace of the system.

5.1 Planar Surface without ROI

Figure 12 shows an image being displayed over the planar surface without any keystone distortion or bleeding. As ROI is absent, the system utilizes whole planar surfaces to display the image. The system also effectively forms a correct projection over the planar surface even if it is moved up to an extent. The orientation of the surface has also changed for various instances. This difference can be observed clearly in Figures 12, 13, 14, and 15.

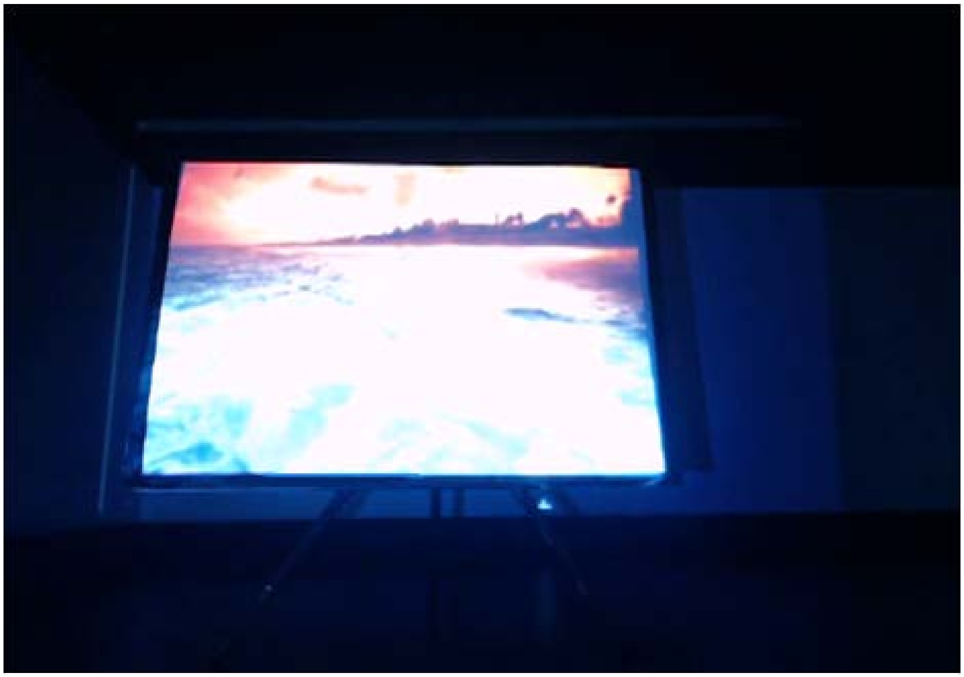


Figure 12: Planar surface is used a display screen, while ROI is not provided. The projection is keystone corrected and does not bleed out of the planar surface

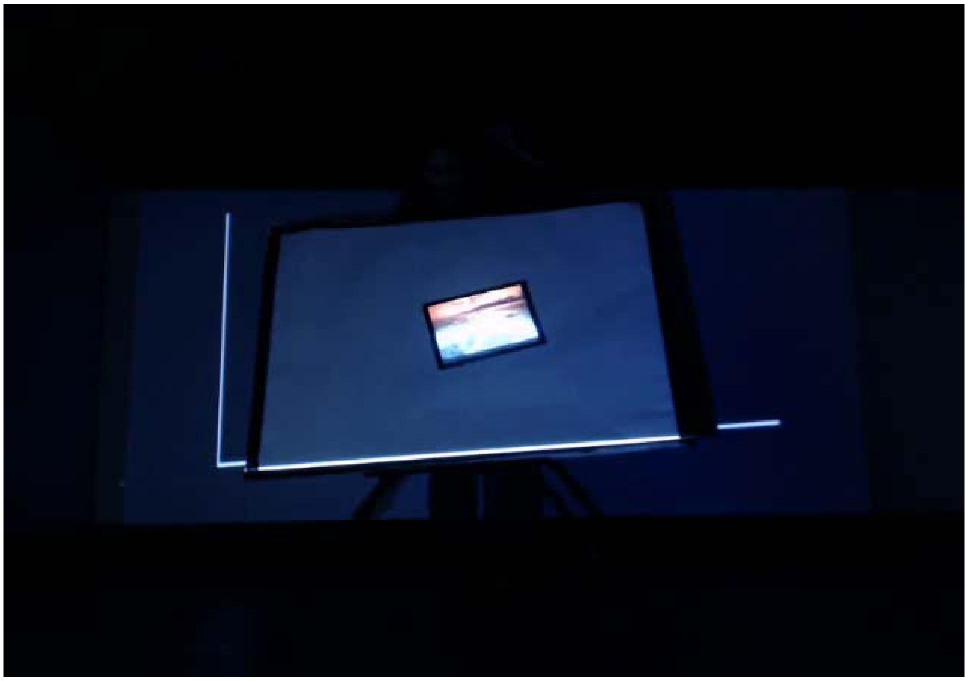


Figure 13: Projection fits the rotated rectangular ROI with proper orientation and position precisely. The illuminated lines are boundaries of window displaying the image

5.2 Planar Surface with ROI

The projection image modifies as soon as the ROI is moved and fits over it correctly provided that the ROI is moved with a constant speed and there are no sudden changes or movements. If the ROI is moved abruptly, the projection follows it, and then oscillates about it while the distance error between the current position and the required position decreases. The oscillations of the projection finally converge to the correct position of the ROI for all the instances.

Figures 14 and 15 illustrate the occasions where the system has successfully formed a precise projection over a triangular and a circular ROI, respectively. For the case where the ROI is abruptly moved from an extreme corner to the other extreme corner, the setting time or convergence time for the projection is calculated to be 5 seconds on the average.

Figure 13 shows the projection over rectangular ROI, where the projection image can be seen perfectly fitted over the ROI, even though the ROI has significantly larger angle of rotation. The illuminated lines in Figures. 13, 14, and 15, are the boundaries of the window containing the rendered image. The resolution of the rendered image can also be adjusted to eliminate these lines; however, the concern of this application is not affected by them.

During these experiments, we moved the ROI abruptly to test its efficiency. The system successfully fits the projection after a few oscillations upon abrupt motion. It is also moved with a constant speed while the projection fits over it perfectly.

6 Discussion

The results of this approach can be improved by introducing fast computing techniques such as the use of GPU and a camera with a better frame per second rate. As we used a webcam, an addition of a camera with better resolution will provide considerably precise information.

The accuracy of the system can yet be improved by taking more images while forming homography mappings. The homography with a least re-projection error can be selected from the homographies calculated using these images. However, these benefits come with a significant cost of either time or expenditure.

A cell phone application version of this system is also being considered to be designed. The cell phone application will utilize the in-built camera of the cell phone and a Pico projector will be required, to be connected with the cell phone via a data cable. Such a system will have an exceptional portability, as cell phones and Pico projectors are sufficiently compact and easily portable.

7 Conclusion

We have designed a simple system for projection over a mobile region of interest by using an uncalibrated camera. The system tracks and projects over the region of interest with true orientation and scale, while the region of interest moves with a certain constant speed. It takes minimal information to form

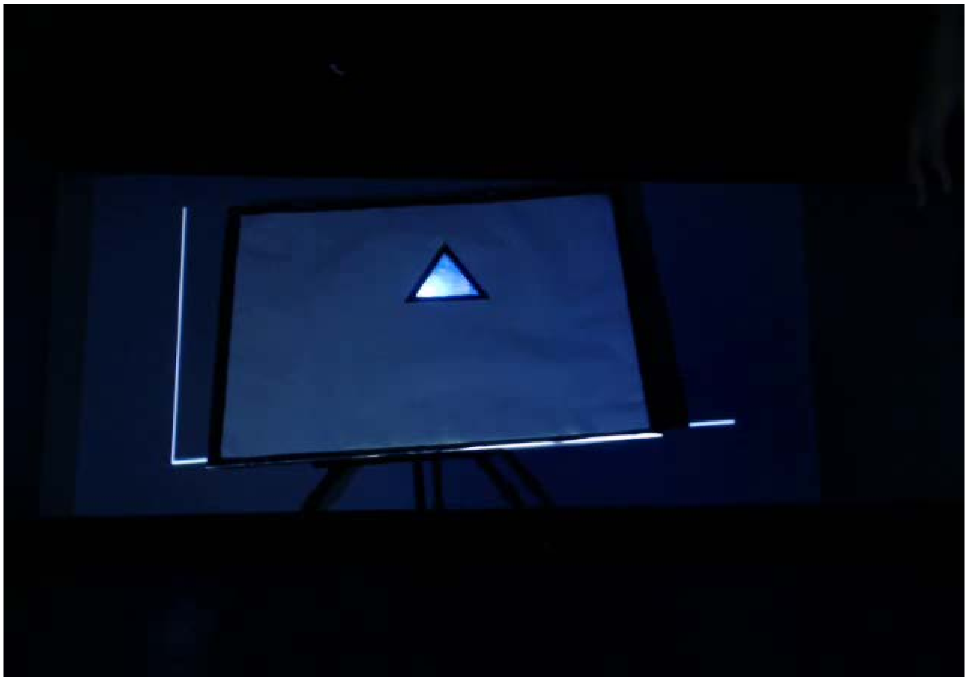


Figure 14: Triangular ROI is introduced over the planar surface. The system identifies the shape and projects over it, while it moves in the planar surface

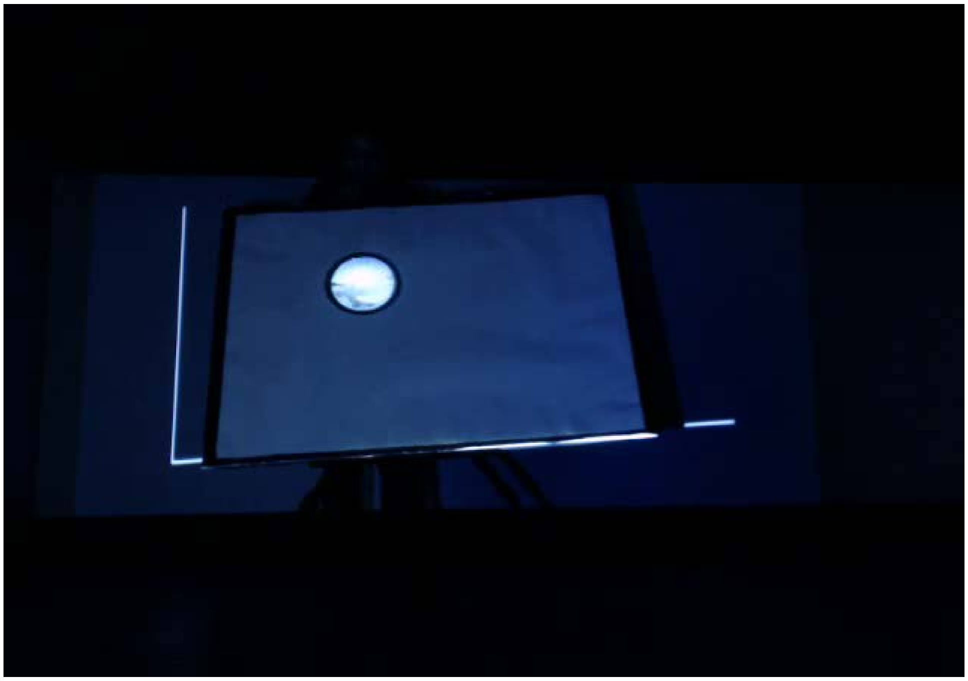


Figure 15: The instance of circular ROI with image projected over it. The projection for this complex shaped ROI overlays entirely without any bleeding

homographies between a projector, an uncalibrated camera, and the planar surface with ROI in it. The system is provided with no information about the intrinsic and extrinsic parameters and neither calculates them. It is assessed to be efficient enough for projection over the ROI without bleeding out of it. As other researches depend over specialized equipment to track the mobile projection area, our system excludes all such specialized apparatuses and equipment. Our system can also utilize the in-built camera of the laptop or even of a cell phone for this purpose, making it suitable for common use.

References

[1] D. Badyopadhyay, R. Raskar, and H. Fuchs, "Dynamic Shader Lamps: Painting on Real Objects", IEEE ACM Int. Symposium on Augmented Reality (ISAR), New York, 2001.

[2] M. Beus, R. Blach, S. Stegmaier, and U. Hafner, "Towards a Scalable High-Performance Application Platform for Immersive Virtual Environments", Eurographics Workshop on Virtual Environments, 2001.

[3] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. Kenyon, and J. C. Hart, "The CAVE, Audio Visual Experience Automatic Virtual Environment", Communications of the ACM, pp. 64-72, 1992.

[4] P. V. C. Hough, "Machine Analysis of Bubble Chamber Pictures", *Proc. Int. Conf. High Energy Accelerators and Instrumentation*, pp. 554-558, 1959.

[5] M. K. Hu, "Visual Pattern Recognition by Moment Invariants", *IRE Transactions of Information Theory*, pp. 179–187, 1962.

[6] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", *Transactions of the ASME – Journal of Basic Engineering*, pp. 34-45, 1960.

[7] J. C. Lee, P. H. Dietz, D. Aminzade, and S. E. Hudson, "Automatic Projector Calibration using Embedded Light Sensors", *Proc. ACM UIST '04*, pp. 123-126, 2004.

[8] J. C. Lee, S. E. Hudson, J. W. Summet, and P. H. Dietz, "Moveable Interactive Projected Displays Using Projector Based Tracking", *Proc. ACM UIST '05*, pp. 63-72, 2005.

[9] T. Okatani, and K. Deguchi, "Autocalibration of a Projector-Camera System", *IEEE Transactions of Pattern Analysis and Machine Intelligence* 27(12):1845-1855, 2005.

[10] U. Ramer, "An Iterative Procedure for the Polygonal Approximation of Plane Curves", *Computer Graphics and Image Processing* 1(3):244-256, 1972.

[11] R. Rasker, M. S. Brown, R. Yang, W. C. Chen, G. Welch, and H. Towles, "Multi-Projector Displays Using Camera-Based Registration", *Proc. IEEE Visualization*, pp. 161-168, 1999.

[12] R. Raskar, P. A. Beardsley, "A Self-Correcting Projector", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), pp. II-504-II-508 2001.

[13] R. Raskar, P. A. Beardsley, J. V. Baar, Y. Wang, P. H.

Dietz, J. Lee, D. Leigh, and T. Willwacher, "RFIG Lamps: Interacting with a Self-Describing World via Photosensing Wireless Tags and Projectors", *Proc. of ACM SIGGRAPH*, 23(3):406-415, 2004.

[14] R. Suthankar, R. Stockton, and M. Mulin, "Automatic Keystone Correction for Camera-Assisted Presentation Interfaces", *Proc. International Conference on Multimodal Interfaces (ICMI)*, pp. 607-614, 2000.

[15] R. Suthankar, R. Stockton, and M. Mulin, "Smarter Presentations: Exploiting Homography in Camera-Projector Systems", *Proc. International Conference on Computer Vision*, pp. 247-253, 2001.

Mamona Awan received Bachelor's degree in Mechatronics and Control Engineering from University of Engineering and Technology, Lahore Pakistan in 2013. She did Masters in Mechatronics from Gwangju Institute of Science and Technology, South Korea in 2016. She has also worked as a research assistant in School of Mechatronics (GIST, South Korea). She is currently a lecturer in National university of Emerging Sciences-FAST, Lahore Pakistan. She has research experience in Computer Vision, Computer Graphics and Spatial Augmented Reality. Her research interests include computer vision, graphics, augmented reality, 3D modeling and robotics.

Kwang Hee Ko received a B.S degree in Naval Architecture and Ocean Engineering from Seoul National University in 1995, M.S. degrees in Mechanical and Ocean engineering in 2001 and a Ph.D. degree in Ocean Engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2003. His research interests include automation of ship and offshore fabrication, feature matching, landmine detection, geometric modeling, CAD/CAM and computer graphics. He worked as a postdoctoral associate at MIT from 2003 to 2004 and as a research associate at the Design and Manufacturing Institute, Stevens Institute of Technology, Hoboken, NJ, from 2004 to 2005. He joined the School of Mechanical Engineering, Gwangju Institute of Science and Technology, Gwangju, Korea, in 2006 and worked as an assistant professor until 2010 and as an associate professor until 2016. He is currently a professor at the Gwangju Institute of Science and Technology, Gwangju, Korea.

CMML: A Cloud Metering Markup Language

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

Abstract

Cloud computing is about utility computing achieved through resource consolidation shared among different applications transparently. Cloud resources are shaped based on the target services provided. A cloud metering framework, that can shape with the cloud resources, need to be in place to be able to meter the cloud resources accurately. The target cloud metering framework needs to be extensible, programmable, scalable, and shareable. At the heart of the proposed framework is an interpreted extensible object oriented cloud metering markup language (CMML). CMML is capable of modeling cloud metering data dynamically and adapt to the elasticity and the scalability of the target cloud environment. A shareable model is continuously maintained by the deployed CMML framework that is capable of modeling cloud metering data in the form of Cloud Metering Objects (CMOs), as well as the cloud metering architecture. The main contribution of this paper is to provide a formal specification of the transactional aspect of CMML through a Structural Operational Semantics (SOS) approach based on the Big-Step and the Small-Step methods.

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 the realization of utility computing. A hybrid pool of resources is managed by the cloud middleware and shapes them dynamically to provide different isolated services. 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. With virtualization being introduced, more complex resources can be constructed, which need to be metered.

A cloud metering framework that can shape accordingly with the cloud services' needs at runtime needs to be in place to accurately and reliably meter the target cloud services. Being shareable and extensible, the metering framework can cater to different applications such as billing, Service Level Agreement (SLA) monitors, Quality of Service (QoS) monitors, predictive resource scaling, etc.

The Cloud Metering Markup Language (CMML) is an object oriented interpreted modeling language that is designed to maintain a shareable cloud metering data model. CMML provides the capabilities, through built-in metering constructs, of correlating resources usage across different architectural layers. Consequently, different metering abstraction levels can be achieved through the flexibility of writing code. The main contribution of this paper is to present the details of the language, and provide a semantic formal specification for the transactional perspective of the language. More information about the whole framework is presented in [25].

In Section 2 we present the background followed by related work in section 3. We present the problem characteristics in section 4 and introduce the proposed metering framework in section 5. In Section 6, we present the formal specifications of the CMML language, and we conclude in section 7.

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.

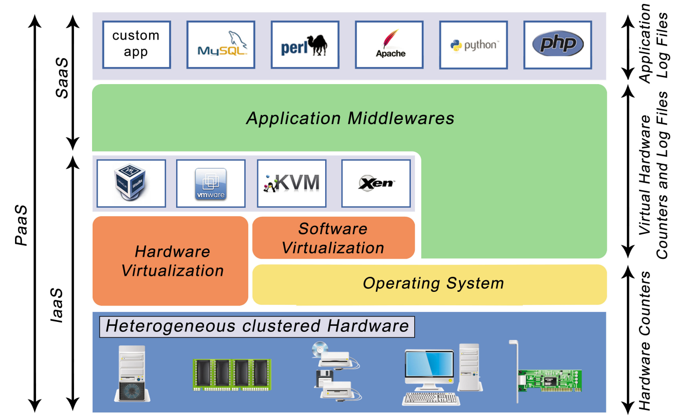


Figure 1: Cloud architectural layers

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. 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

---

[*] 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.

ISCA Copyright© 2017

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.

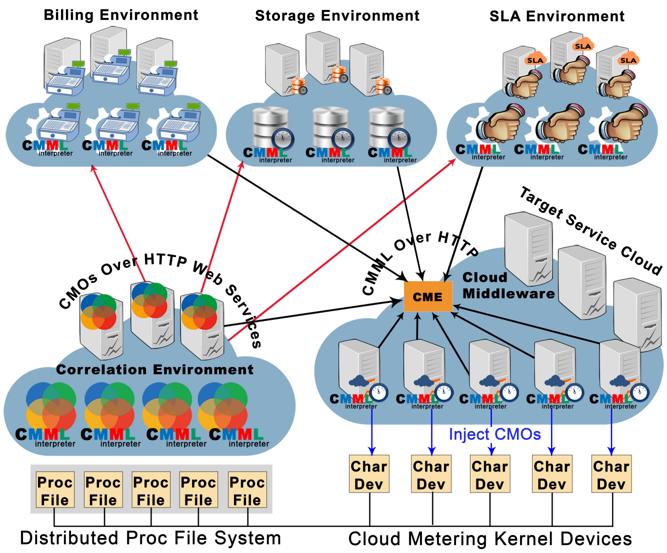


Figure 2: Multi-tier metering architecture

The phases of the metering process 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 [12] 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 [24]. A comprehensive study for power consumption in data centers by Anton Beloglazov et al. is presented in [5]. 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 [20] 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 [22]. 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 [9] 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 [8] through automatic log evaluation based on XML. Predictive Modelling Markup Language (PMML) is presented by Guazzelli et al. in [7] 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 [27] for correlating distributed logs, and using dispatchers for load balancing and scalability. William M. Jones et al. presented analytical and simulation-based approaches in [10] showing the negligible impact of choosing a sub-optimal checkpoint. The issues of continuous sampling are raised by Gang Ren et al. in [20]. 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, 15, 16] for a cloud accounting system and charging policy based on a domain specific language (DSL). The DGAS, Distributed Grid Accounting System, is presented in [17] as an accounting infrastructure for grid environments. The GridBank (GB) is presented in [4] 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 [14], 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, license 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

Cloud metering inherits its complexity from that of cloud environments. The target metering framework 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 services running on the cloud, 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.

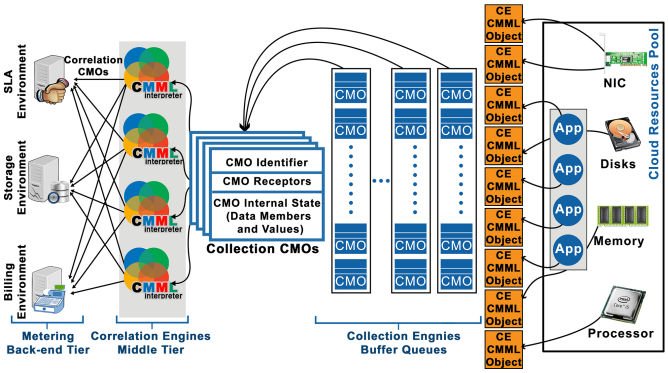


Figure 3: Metering framework architecture

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 requirement. 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 supported, 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 mandates 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 with 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.

A full prototype has been built for the proposed unified cloud metering framework and was applied on a real life cloud-deployed online shopping store environment as a case study for performace evaluation. The details of the framework design, prototype, case study, deployment decisions, ANOVA/GLM experiments design and results are presented in [25].

6 Formal Specifications

In this section a formal specification for CMML is presented following the Structural Operational Semantics (SOS) approach [18][19], coupled with a syntactical set notation specifications. This allowed us to show the validity of the language's operational aspect, as well as providing a formal specification for the language syntax. We have decided to concentrate on SOS for numerous reasons. Since the functional and operational aspects are the main contribution of CMML over other markup languages, we have chosen SOS over Denotational and Axiomatic methods. The drawbacks of using a purely denotational definition in a context like ours are enumerated by G. Kahn in [11]. Moreover, flattening all expressions in CMML as markup emphasised withdrawing the denotational approach. On the other hand, Axiomatic methods such as Hoare Logic [3], are concerned with a specific program correctness and not with the general semantics of the whole language. Finally, CMML inherits its syntax from SGML [26], which makes it effeciently extensible[21], and allowed us to take that fact for granted.

6.1 Set Notation Formal Specifications

In this section we will present a set of rules that defines the general syntax of the language. Listing 3 presents a primitive set that we will use in our rule definitions.

```
lc = [a  z]   The set of all lower case characters
uc = ['A  Z]   The set of all upper case characters
dig = [0  9]   The set of all digits
pascii = [' '  ' ~'] The set of all printable ASCII character
      from ASCII(32) to ASCII(126)
```

Listing 3: Primitive sets

CMML has a basic set of predefined CMML tags, which are the minimum set of tags that need to be supported for a CMML script to execute correctly. Listing 4 presents the CMML mandatory tags.

```
CMMLScriptTag = {"CMMLScript"}
CMMLMainTag = {"CMMLMain"}
CMMLReservedTags ={"CMMLRoutine","CMMLInclude","
      CMMLRemoteInclude","CMMLClass","CMMLObject", "Exec"}
CMMLInternalTags = {"Subject","Value","Name"}
DataTypes = {integer,boolean,double,float,long,string,
      CMMLObject,numeric}
Visibility = {Public,Private}
Boolean = {True,False}
```

Listing 4: Predefined CMML tags

A CMML Script is defined based on CMML tags that enclose both data and operational logic. As described formally in listing 5, a CMML tag is a tag whose name should start with "CMML" and is a maximum of 80 characters. A CMML tag has the following formal definition.

```
CSTN: CMML Service Tag Name
CSTN = {s | s is a string such that
               s ∈ "CMML"(lc|uc)n and n > 0 and n <=76
                and s ∉ CMMLScriptTag ∪ CMMLMainTag ∪
                     CMMLReservedTags}
```

Listing 5: CMML service tag name

For the sake of simplicity and encapsulation, we define in listing 6 a set of functions that return sets of elements that we will refer to in our subsequent definitions.

```
MethodNames(β) = The set of method names of the CMML Object
      β or CMML Class β based on the context
MethodParameters(β,δ) = The set of parameter names of the
      method δ of the CMML Object β or CMML Class β based on
      the context
DataMembers(β) = The set of data member names of the CMML
      Object β or CMML Class β based on the context
```

Listing 6: Set functions

Listing 7 presents the sets of tags following common patterns. We will refer to those sets in the our definitions of more complex tags.

```
GT: General Tag
GT = {s | s is a string such that s ∈ (lc|uc)n where n > 0
      and n <=80 }

STR: Simple Tag Record
STR = {s | s is a string such that s ∈ "<"β">"pascii+"</"
      β">" and β ∈ GT}

CTR: Composite Tag Record
CTR = {s | s is a string such that s ∈ "<"β"><Name>"δ"</
      Name>"(CTR|STR)+"</"β">"
and β ∈ GT and δ ∉ TagNames, and δ will be added to
      TagNames after successful declaration}

CSR: CMML Service Record
CSR= {s | s is a string such that s ∈ "<"β">"(STR|CTR)+"</"
      β">" and β ∈ CSTN}
```

Listing 7: CMML different tag types definition

The CMML language is an extendable language in the sense that it can be extended by adding new tags to it. Within our scope we will not be able to define each and every CMML tag currently in the language as they follow the operational definition behind the need of their functionality; a tag is added for a specific functionality whose need arises due to its absence. So a special set notation definition can be constructed as per CMML tag, but the most important matter is that it needs to be a subset of the general definition of the CSR tag defined above; CMML Service Record. Consequently, we will choose a set of complex fundamental CMML tags and present their set notation specification as examples, and similarly other CMML tag definitions can follow the same line of definition.

The Exec Tag: The Exec tag is a special fundamental tag used to invoke any tag declared in the current CMML Script by name. Listing lst:CMML Exec Tag defines the Exec tag.

```
Exec = {s | s is a string where
                s ∈ "<Exec>"(β|CSR)"</Exec>" and β ∈ TagNames
                    }
```

Listing 8: CMML Exec tag

The CMMLMain Tag: The CMMLMain tag, defined in listing lst:CMML Main Record, should be located inside the CMMLScipt tag only once and it designates the starting point of execution of the script.

```
CMR: CMML Main Record
CMR = { s | s is a string such that s ∈ "<"β">"(CSR ∪ EXEC)
      * "</"β">"
           and β ∈ CMMLMainTag}
```

Listing 9: CMML main record

The CMMLScript Tag: The CMMLScript tag, defined in listing 10, is the main tag that defines a CMML script and it encloses all its CMML tags. It essentially needs the CMMLMain tag to be defined some where to designate the starting point of the execution.

```
CSCR: CMML Script Record
CSCR = {s | s is a string where s ∈ "<"β">"(CSR|CMMLClass)*
      CMR "</"β">" and β ∈ CMMLScriptTag }
```

Listing 10: CMML script record

The CMMLClass Tag: The CMMLClass tag, defined in listing 11, is the most complex tag in the language as it defines an object oriented class. The CMMLClass tag encloses all the class definitions including data members, methods, and metering constructs.

```
CMMLClass = { s | s is a string such that
     s ∈ "<CMMLClass>
    <Name>"σ"</Name>"
    (ε |"<FlattenedName>"μ"</FlattenedName>")
    "<DataMembers>"
      (ε |"<DataMember>"
        "<Name>"δ"</Name>"
        <Visibility>"β"</Visibility>"
        <Type>"κ"</Type>"
        "<Exportable>"λ"</Exportable>"
        "<Sync>"ζ"</Sync>"
        "<Billing>"ζ"</Billing>"
        "<SLA>"ζ"</SLA>"
        "<Size>"dig+"</Size>"
        (ε |"<FetchScopes>"
          ("<FetchScope>"pascii+"</FetchScope>")+
        </FetchScopes>")*)*
    "</DataMembers>"
    "<Collect>"(CSR ∪ Exec)*"</Collect>"
    "<Correlate>"(CSR ∪ Exec)*"</Correlate>"
    "<Bill>"(CSR ∪ Exec)*"</Bill>"
    "<SLA>"(CSR ∪ Exec)*"</SLA>"
    "<Methods>"
      ("<Method>"
        "<Name>"ω"</Name>"
         (ε | "<Parameters>"
          ("<Parameter>
            <Name>"ϕ"</Name>
            <Type>"ς"</Type>
          </Parameter>")+
        "</Parameters>")
        "<Body>"
          "<CMML>"(CSR ∪ Exec)*"</CMML>"
        "</Body>"
      "</Method>")*

    "</Methods>"
    "</CMMLClass>"
    and σ ∈ pascii+ and σ ∉ TagNames
    and μ ∈ pascii+ and μ ∉ TagNames ∪ {σ}
    and δ ∈ pascii+ and δ ∉ DataMembers(σ)
    and β ∈ Visibility and κ ∈ DataTypes
    and λ ∈ Boolean and ζ ∈ Boolean
    and δ ∈ pascii+ and δ ∉ DataMembers(σ)
    and ω ∈ pascii+ and ω ∉ MethodNames(σ)
    and ϕ ∈ pascii+ and ϕ ∉ MethodParameters(σ,ω)
    and ς ∈ DataTypes}
```

Listing 11: CMMLClass tag definition

The CMMLExecuteMethod Tag: To execute a method of an instantiated object, the CMMLExecuteMethod tag, defined in listing 12, is invoked with the target object reference and the method name as well as parameters.

```
CMMLExecuteMethod = { s | s is a string where
     s ∈ "<CMMLExecuteMethod>
    <CMMLObject>"pascii+"</CMMLObject>
    <CMMLObjectMethod>"pascii+"</CMMLObjectMethod>"
     ( ε | "<Parameters>"
      ("<Parameter>
        <Name>"pascii+"</Name>
        <Value>"pascii+"</Value>
      </Parameter>")*
    "</Parameters>")
    "</CMMLExecuteMethod>" }
```

Listing 12: CMMLExecuteMethod tag definition

6.2 Operational Semantics

In this section we will present the the Big-Step [11] and the Small-Step [13] SOS methods. We chose three fundamental operations to illustrate, namely expression evaluation, loop iterations, and conditionals. After presenting the Big-Step proofs, we will show their deatiled breakdown derivations using the Small-Step method.

6.2.1 Big-Step Semantics

Expression Evaluation: The CMMLAdd and the CMMLIncrement are two tags that perform mathematical operations. The CMMLAdd adds any number of operands and returns the result of the summation to the caller tag, and the CMMLIncrement increments a value with an offset and returns the new value to the caller tag. The example in listing 13 illustrates both in one shot.

```
<CMMLAdd>
 <Value>
  <CMMLIncrement>
   <Value>10</Value>
   <Inc>3</Inc>
  </CMMLIncrement>
 </Value>
 <Value>12</Value>
</CMMLAdd>
```

Listing 13: Expression evaluation example

The CMMLAdd can take any number of the “Value” tag and sum their values. The “Value” tag can enclose either a constant or another expression. In case of another expression, the expression will need to be evaluated first and the result will be used in the summation. The CMMLIncrement has two tags, the first one is the “Value” tag and it should be incremented by the value of the “Inc” Tag. Similarly, the “Value” and the “Inc” tags of the CMMLIncrement tag can enclose either constants or an expression that will need to be evaluated first before the CMMLIncrement can perform its operation.

The CMML Tags were broken down to represent its internal tags, so we have introduced three new tags to be used in the Big-Step Semantics which are Add-Value, Increment-Value, and Increment-Inc. The three tags identify the starting point of the internal tag to represent the substitution of their internal enclosed values. Figure 4 shows the Big-Step semantics for both CMMLAdd and CMMLIncrement.

$$Add-Value\,\frac{true}{n \Downarrow n} \tag{1}$$

$$Increment-Value\,\frac{true}{n \Downarrow n} \tag{2}$$

$$Increment-Inc\,\frac{true}{n \Downarrow n} \tag{3}$$

$$Add\,\frac{VT_i \Downarrow n_i}{\Sigma_{i=1}^{k} VT_i \Downarrow n_{total} \quad where \quad n_{total} = \Sigma_{i=1}^{k} n_i} \tag{4}$$

$$Increment\,\frac{VT \Downarrow n_1 \qquad VI \Downarrow n_2}{VT + VI \Downarrow n_{result} \qquad n_{result} = n_1 + n_2} \tag{5}$$

Figure 4: Big-Step expression evaluation

The CMMLAdd-Value, CMMLIncrement-Value, and the CMMLIncrement-Inc tags are reduced to the constant value of the equivalent expression that they enclose. CMMLAdd is reduced to the summation of all the return values of all the CMMLAdd-Value tags, and the CMMLIncrement is reduced to the summation of the return values of the enclosed CMMLIncrement-Value and the CMMLIncrement-Inc tags.

For verification, we apply the above semantics to the example presented earlier in listing 13 to prove the validity of its operations with respect to the CMML syntax as well as the tag semantics. Figure 5 illustrates the proof steps. It is important to highlight that the example proof works in a bottom-up inference approach where the full tag is represented at the lowest level and is broken down until we reach the top; so every atomic tag will be represented by a portion and its yield or reduction will be stated right below it.

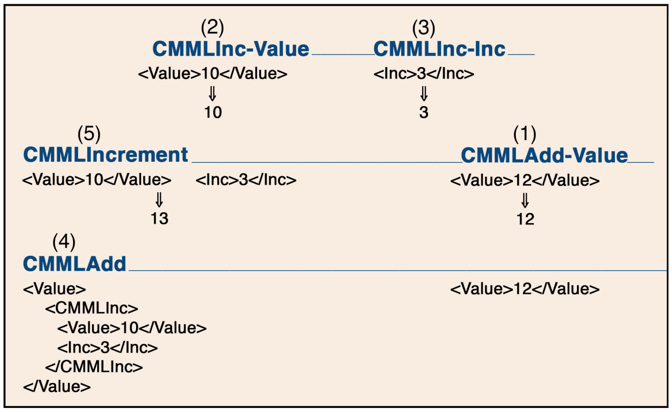


Figure 5: Big-Step expression evaluation proof example

Repeat-Until Loop: The CMMLRepeatUntil tag implements the repeat-until loop. It encloses two tags, the “LoopBody” tag and the “Until” tag. The “LoopBody” tag encloses a sequence of CMML tags to be executed as a routine and is considered the loop body that the loop engine iterates on as long as the expression enclosed in the “Until” tag evaluates to false. Listing 14 shows an example of the CMMLRepeatUntil loop. Basically it loops on the CMML tags that increment an object data member and exits the loop when the data member value is greater than three.

```
<CMMLRepeatUntil>
    <Name>Check_Value</Name>
    <LoopBody>
        <CMMLObjectAssignDataMember>
            <CMMLDataMember>
             int_val
            </CMMLDataMember>
            <AssignTo>
                <CMMLInc>
                    <Value>
                        <CMMLObjectFetchDataMember>
                            <CMMLDataMember>
                              int_val
                           </CMMLDataMember>
                        </CMMLObjectFetchDataMember>
                    </Value>
                    <Inc>1</Inc>
                </CMMLInc>
            </AssignTo>
        </CMMLObjectAssignDataMember>
    </LoopBody>
    <Until>
        <CMMLGreaterThan>
            <Subject>
                <CMMLObjectFetchDataMember>
                    <CMMLDataMember>
                      int_val
                    </CMMLDataMember>
                </CMMLObjectFetchDataMember>
            </Subject>
            <Value>3</Value>
        </CMMLGreaterThan>
    </Until>
</CMMLRepeatUntil>
```

Listing 14: Repeat-Until loop example

Figure 6 shows the Big-Steps semantics of the CMMLRepeatUntil tag. We have constructed three new tags to be able to cover all situations. The first tag is the CMMLRepeatUntil-Out which is a construct indicating the transfer of the execution to the next CMML tag outside the CMMLRepeatUntil tag. The CMMLRepeatUntil-FALSE will be invoked in case the Until tag yields a false value, and the CMMLRepeatUntil-TRUE will be invoked in the case of the Until tag yielding a true value.

$$RepeatUntil \Downarrow RepeatUntil-LoopBody \tag{6}$$

$$RepeatUntil-LoopBody \Downarrow RepeatUntil-Until \tag{7}$$

$$RepeatUntil-Out \Downarrow Next-Instruction \tag{8}$$

$$RepeatUntil-Until\,\frac{true}{n \Downarrow n} \tag{9}$$

$$RepeatUntil-FALSE\,\frac{RepeatUntil-Until \Downarrow false}{RepeatUntil \Downarrow RepeatUntil-LoopBody} \tag{10}$$

$$RepeatUntil-TRUE\,\frac{RepeatUntil-Until \Downarrow true}{RepeatUntil \Downarrow RepeatUntil-Out} \tag{11}$$

Figure 6: Big-Step repeat-until loop semantics

Figure 7 shows a proof of the example introduced in the code listing 14 above using the Big-Step semantics rules of the CMMLRepeatUntil tag. As we can see, the proof inference starts with rule 1 and alternates between rule 2 and 5. Rule 6 is then invoked when the until condition yields false.

Conditional Case Statement: Adopting the same template used in expression evaluation and loop iteration, we illustrate the conditional case statement. The CMMLCase statement is a very extensible CMML conditional tag that can achieve both the if-then-else statement as well as the case-switch statement. The CMMLCase statement can contain an unlimited number of CMMLWhen tags, which is considered a conditional block. The CMMLWhen tag contains three optional tags which are CMMLCondition, Exec, and Otherwise. The CMMLCondition should yield a Boolean true or false value; it should enclose a Boolean or an expression that yields to Boolean. Based on the evaluation of the CMMLCondition tag, the execution will branch. If the CMMLCondition tag yields true the Exec tag will be invoked and the CMMLCase will then exit. On the other hand, if the CMMLCondition yields false, then the interpreter should check if there is an Otherwise tag, in which case it will be invoked or else the execution will be transfered to the next CMMLWhen tag if any exist. The process will continue until a

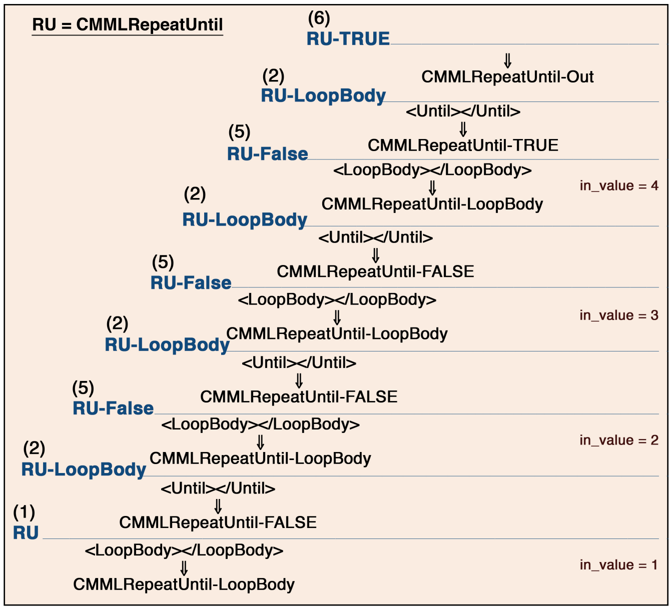


Figure 7: Big-Step Repeat-Until loop proof example

CMMLCondition of a CMMLWhen tag yields true, Otherwise tag is reached and executed, or all the CMMLWhen tags are exhausted.

Listing 15 represents an example of a CMMLCase that checks the value of a data member and prints a message on the standard output accordingly. As we can see from the code, if the data member int_val is equal to 5 then the CMMLCase will print out a message saying that the value is 5, else if it is equal to 10 another message will be printed saying that the value is 10, otherwise a message will be printed indicating that the value is neither 5 nor 10.

```xml
<CMMLCase>
    <CMMLWhen>
        <CMMLCondition>
            <CMMLEqual>
                <Subject>
                    <CMMLObjectFetchDataMember>
                        <CMMLDataMember>
                          int_val
                        </CMMLDataMember>
                    </CMMLObjectFetchDataMember>
                </Subject>
                <Value>5</Value>
            </CMMLEqual>
        </CMMLCondition>
        <Exec>
            <CMMLOut>
                <Subject> int_val = 5 </Subject>
                <Target>
                 <PipeTo>STDOUT</PipeTo>
                </Target>
            </CMMLOut>
        </Exec>
    </CMMLWhen>
    <CMMLWhen>
        <CMMLCondition>
            <CMMLEqual>
                <Subject>
                    <CMMLObjectFetchDataMember>
                        <CMMLDataMember>
                          int_val
                        </CMMLDataMember>
                    </CMMLObjectFetchDataMember>
                </Subject>
                <Value>10</Value>
            </CMMLEqual>
        </CMMLCondition>
        <Exec>
            <CMMLOut>
                <Subject> int_val = 10 </Subject>
                <Target>
                 <PipeTo>STDOUT</PipeTo>
                </Target>
            </CMMLOut>
        </Exec>
        <Otherwise>
            <CMMLOut>
                <Subject> int_val is neither equal to 10 nor 5 </
                      Subject>
                <Target>
                 <PipeTo>STDOUT</PipeTo>
                </Target>
            </CMMLOut>
        </Otherwise>
    </CMMLWhen>
<CMMLCase>
```

Listing 15: Conditional case statement example

Figure 8 presents the Big-Step semantic rules for the CMMLCase. Notice here also that we have the newly defined tag CMMLCase-Out which is used to exit or break the execution of the CMMLCase tag. The CMMLCase-Out simply transfers the execution pointer for the current script to the following CMML tag in the current script's chronological execution order.

$$Case \Downarrow Case_0 \tag{12}$$

$$RepeatUntil-Out \Downarrow Next-Instruction \tag{13}$$

$$Case-TRUE_i \frac{Count(When) > i}{When_i} \tag{14}$$

$$Case-FALSE_i \frac{Count(When) <= i}{Case-Out} \tag{15}$$

$$When-TRUE-Exec_i \frac{Condition_i \Downarrow true \quad !Exist(Exec_i)}{Case-Out} \tag{16}$$

$$When-TRUE \frac{Condition_i \Downarrow false \quad !Exist(Exec_i)}{Case-Out} \tag{17}$$

$$When-FALSE-Otherwise_i \frac{Condition_i \Downarrow true \quad !Exist(Otherwise_i)}{Case-Out} \tag{18}$$

$$When-FALSE_i \frac{Condition_i \Downarrow false \quad !Exist(Otherwise_i)}{Case_{i+1}} \tag{19}$$

Figure 8: Big-Step case statement specifications

For the CMMLCase tag we will show three proofs of the example presented in the above code snippet presented in code listing 15. The three presented proofs correspond to the tree cases attempting to cover all the alternative execution paths. Figure figures 9 shows the execution path when the int_value is equal to 5 which will results in executing the CMML code within the “Exec” tag of the first “CMMLCondition” of the first “CMMLWhen” tag. Figure 10 shows the execution path when the int_value is equal to 10 and hence executing the “Exec” tag of the second “CMMLWhen”. Finally, figure 11 shows the execution path when int_value is equal to any other value which results in executing the CMML “Otherwise” tag.

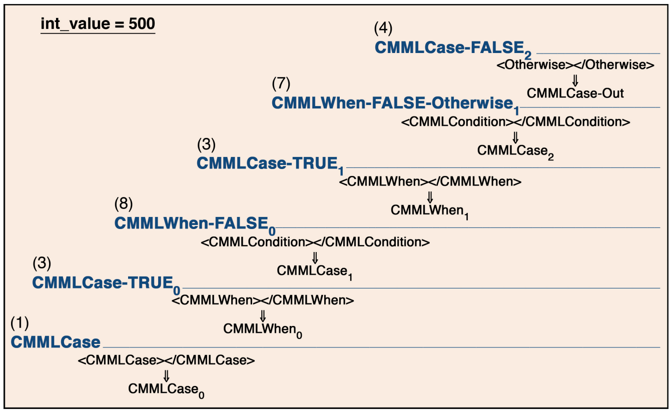


Figure 9: Big-Step conditional case statement proof example

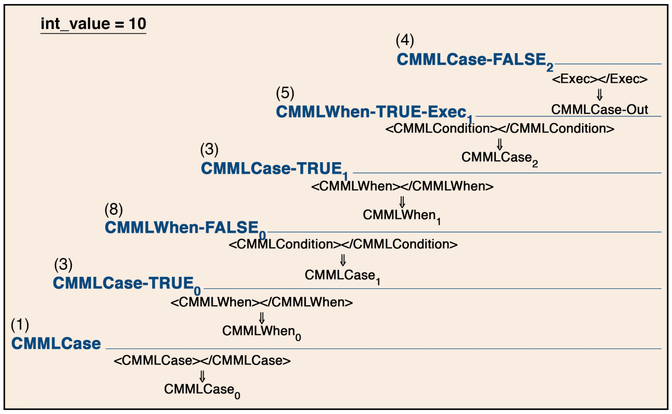


Figure 10: Big-Step conditional case statement proof example

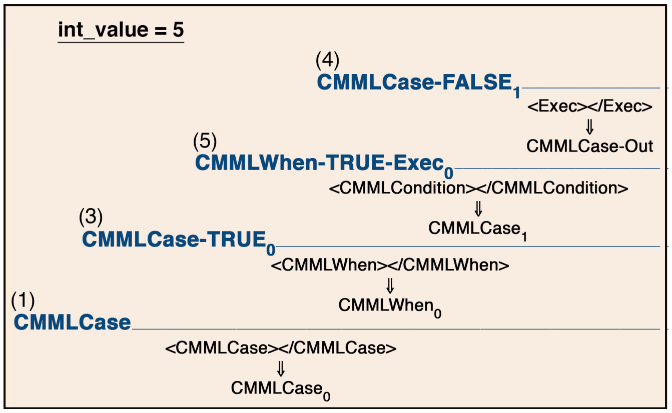


Figure 11: Big-Step conditional case statement proof example

6.2.2 Small-Step Semantics In the Small-Step semantics we will break down the Big-Step Semantics illustrated in the previous section to show the details of execution of the three fundamental operations. The importance of the Small-Step semantics is that it gives an insight of how expressions are evaluated, how loops iterate, and how conditionals perform branching decisions.

To be able to present the Small-Step semantics, we need to define a state that needs to be maintained by the CMML tag execution engine, and which represents the internal state of the interpreter during execution in each step of inference. Also the state will be passed to sub-tags that need to execute recursively within the context of a running tag. Listing 16 shows an abstract structure that represents the state of the CMML tag during execution.

```
struct state{
       int i;
       float f;
       long l;
       double d;
       string s;
       bool b;
       bool intFlag;
       bool floatFlag;
       bool longFlag;
       bool doubleFlag;
       bool stringFlag;
       bool boolFlag;

       params: a key-value array of parameters declared for the
             tag;
       currentParam: a pointer to the current param item in the
             params array
       parentStatus: a pointer to the parent tag state struct
};
```

Listing 16: CMML tag state structure

The state has a set of primitive type fields and a set of corresponding flags indicating which one is set. We will assume in our derivations below that when a primitive type variable is set, its corresponding internal flag will be set automatically. Three other important variables are defined which are the params, currentParam, and parentStatus. In any CMML tag, each enclosed tag should be evaluated to a value based on its enclosed content. The params is a vector of a set of elements equivalent to the number of tags inside the current CMML tag, and each element of the params vector represents the value of the tag that results after evaluating the tag. The currentParam is an index defining which element in the parameters vector is currently being executed. Finally, the parentStatus is a pointer that points to the state variable of the parent CMML tag.

To be able to construct the Small-Step semantics for CMML, we have extended the tags with extra sub-tags to identify the opening, the closing, and the body of each tag, e.g. the Value tag of the CMMLAdd will be decomposed to CMMLAdd-ovalue, CMMLAdd-cvalue, and CMMLAdd-value respectively. This will allow us to describe the behavior of the language in each atomic step of execution. To be able to handle tag recurrence, we use a subscript to identify the sequence of the tag based on its appearance, e.g. in case of the CMMLAdd we can have an unlimited number of "Value" tags, and hence we can have CMMLAdd-ovalue$_i$ to designate a specific tag. Finally, for the sake of the diagrams clarity, we will use abbreviations for long CMML tags like CMMLRepeatUntil and CMMLCondition to avoid diagram congestion and achieve clear and easy to read derivation diagrams.

A rule in the small-step semantics has three constructs, namely: a name, a condition, and an execute statement. The name identifies the entry point of the rule. A rule can only be executed if its condition yields true. The execute statement is an implication statement that describes the operations to be performed when the target rule fires. Multiple rules with the same name can exist for different conditions. Figure 12 represents a general template for a small-step rule.

$$Name \frac{Condition}{< SourceTag, SourceState > \rightarrow < ReducedTag, NewState >} \tag{20}$$

Figure 12: Small-Step specification rule

The execution rule is in the form of a reduction from one state to another. The source tag and state are reduced to a new tag and state. Throughout rules traversal, the reduced tag is used to fetch the next rule to be inferenced until the execution terminates via a predefined implicit tag that has an "Exit" postfix in its name, e.g Add-Exit.

Expression Evaluation: Figure 13 illustrates the small-step semantics of the CMMLAdd tag demonstrating the internals of its execution. Upon arriving at a CMMLAdd tag, rule 1 is invoked identifying the entry point for the CMMLAdd tag. The state of the tag is initialized, and the parent state passed to the tag will be saved in the state ParentStatus variable. Rule 2 and 3 have the same name but with different preconditions. Rule 2 captures the end of execution of the tag while rule 3 performs a transition to the next "Value" to be processed. As long as there are still "Value" tags to be processed the inference engine will iterate over the rule set from 3 to 6. Upon processing the last "Value" tag, rule 2 will be executed leading to the Add-close rule which will reduce to the Add-Exit and lead to the termination of the tag execution. Notice that in rule 7 the state of the execution tag is assigned to the ParentStatus currentParam variable which returns the result of the tag execution to the parent tag.

$$Add-open \frac{i=0}{\begin{array}{l} < Add-open_0, ps > \rightarrow \\ \quad < Add-ovalue_0, s.init(); s.ParentStatus = ps > \end{array}} \tag{21}$$

$$Add-open_i \frac{count(Value) <= i}{< Add-open_i, s > \rightarrow < Add-close, s >} \tag{22}$$

$$Add-open_i \frac{count(value) > i}{\begin{array}{l} < Add-open_i, ps > \rightarrow \\ \quad < Add-ovalue_i, s.init(); s.ParentStatus = ps > \end{array}} \tag{23}$$

$$Add-ovalue_i \frac{true}{< Add-ovalue_i, s > \rightarrow < Add-value_i, s >} \tag{24}$$

$$Add-value_i \frac{true}{\begin{array}{l} < Add-value_i, s > \rightarrow \\ \quad < Add-cvalue_i, params[value_i] = s.currentParam > \end{array}} \tag{25}$$

$$Add-cvalue_i \frac{true}{\begin{array}{l} < Add-cvalue_i, s > \rightarrow \\ \quad < Add-open_{i+1}, s.i+ = params[value_i] > \end{array}} \tag{26}$$

$$Add-close \frac{true}{\begin{array}{l} < Add-close, ps > \rightarrow \\ \quad < Add-Exit, s.ParentStatues.currentParam = s.i > \end{array}}. \tag{27}$$

Figure 13: Expression evaluation Small-Step specifications

Repeat-Until Loop: Figure 14 presents the small-step rules for the CMML conditional statement. The repeat-until loop is split into two main sections, the loop section and the exit condition checking section. Rule 1 will be invoked upon encountering a CMMLRepeatUntil tag, where the state of the tag will be initialized and the ParentStatus will be set to the passed ps state parameter. Rules 2-4 will be executed for the loop body, followed by rules 5-7 for the condition checking. In rule 7 the result of the condition checking is stored in the state boolean attribute to be checked and based on its value either rule 8 or rule 9 is invoked. Rule 8 is invoked if the Until condition evaluates to false which will start the loop body again. Rule 9 is invoked if the Until condition evaluates to true indicating the end of the loop, and invoking the RU-Exit to terminate the tag inferencing execution.

$$RU-open \frac{true}{\begin{array}{l} < RU-open, ps > \rightarrow < Ru-LoopBody, \\ \quad s.init(); s.ParentStatus = ps > \end{array}} \tag{28}$$

$$RU-LoopBody-open \frac{true}{\begin{array}{l} < RU-LoopBody, s > \rightarrow \\ \quad < RU-Until-value, s > \end{array}} \tag{29}$$

$$RU-LoopBody-value \frac{true}{\begin{array}{l} < RU-LoopBody-value, s > \rightarrow \\ \quad < RU-LoopBody-close, \\ \qquad params[loopBody] = s.currentParam > \end{array}} \tag{30}$$

$$RU-LoopBody-close \frac{true}{\begin{array}{l} < RU-LoopBody-close, s > \rightarrow \\ \quad < RU-Until-open, s = s.currentParam > \end{array}} \tag{31}$$

$$RU-Until-open \frac{true}{\begin{array}{l} < RU-Until-open, s > \rightarrow \\ \quad < RU-Until-value, s > \end{array}} \tag{32}$$

$$RU-Until-value \frac{true}{\begin{array}{l} < RU-Until-value, s > \rightarrow \\ \quad < RU-Until-close, \\ \qquad params[Until] = s.currentParam > \end{array}} \tag{33}$$

$$RU-Until-close \frac{true}{\begin{array}{l} < RU-Until-close, s > \rightarrow \\ \quad < RU-close, s.b = params[Until] > \end{array}} \tag{34}$$

$$RU-close \frac{s.b == false}{< RU-open, s > \rightarrow < RU-LoopBody-open, s >} \tag{35}$$

$$RU-close \frac{s.b == true}{\begin{array}{l} < RU-open, s > \rightarrow \\ \quad < RU-Exit, s.ParentStatus.currentParam = s > \end{array}} \tag{36}$$

Figure 14: Repeat-Until loop Small-Step specifications

Conditional Case Statement: Figure 15 presents the small-step rules for the CMML conditional statement.The conditional case statement is inferenced in the same way as the expression evaluation and the Repeat-Until loop. The condition of each CMMLWhen is inferenced by the rules 4-9, and if the condition yields true the corresponding Exec tag is inferenced through the rules 10-14, otherwise the Otherwise tag is inferenced using the rules 15-17. The inference will iterate through all the CMMLWhen statements by their index and terminate either upon executing the Exec or the Otherwise tag of a CMMLWhen or after exhausting all the CMMLWhen tags.

We have presented the small-step semantics for the above three examples as a representative sample of the most

$$Case-open\,\frac{true}{\begin{array}{r}<Case-open_0,ps>\rightarrow<When-open_0,s.init();\\ s.parentstatus=ps>\end{array}} \tag{37}$$

$$Case-open_i\,\frac{count(When)<=i}{<Case-open_i,s>\rightarrow<When-open_i,s>} \tag{38}$$

$$Case-open_i\,\frac{count(When)>i}{<Case-open_i,s>\rightarrow<When-open_i,s>} \tag{39}$$

$$When-open_i\,\frac{Exist(Cond_i)}{<When-open_i,s>\rightarrow<Cond_i,s>} \tag{40}$$

$$When-open_i\,\frac{!Exist(Cond_i)}{<When-open_i,s>\rightarrow<Cond_{i+1},s>} \tag{41}$$

$$Cond-open_i\,\frac{true}{<Cond-open_i,s>\rightarrow<Cond-value_i,s>} \tag{42}$$

$$Cond-value_i\,\frac{true}{\begin{array}{r}<Cond-value_i,s>\rightarrow<Cond-close_i,\\ params[Cond_i]=s.currentParam>\end{array}} \tag{43}$$

$$Cond-close_i\,\frac{Exist(Exec-open_i)}{\begin{array}{r}<Cond-close_i,s>\rightarrow<Exec-open_i,\\ s=params[Cond_i]>\end{array}} \tag{44}$$

$$Cond-close_i\,\frac{!Exist(Exec-open_i)}{\begin{array}{r}<Cond-close_i,s>\rightarrow<Case-close,\\ s=params[Cond_i]>\end{array}} \tag{45}$$

$$Exec-open_i\,\frac{s.b=true}{<Exec-open_i,s>\rightarrow<Exec-value_i,s>} \tag{46}$$

$$Exec-open_i\,\frac{s.b=falseExist(Otherwise_i)}{<Exec-open_i,s>\rightarrow<Otherwise_i,s>} \tag{47}$$

$$Exec-open_i\,\frac{s.b=false!Exist(Otherwise_i)}{<Exec-open_i,s>\rightarrow<Case_{i+1},s>} \tag{48}$$

$$Exec-value_i\,\frac{true}{\begin{array}{r}<Exec-value_i,s>\rightarrow<Exec-close_i,\\ params[Exec_i]=s.currentParam>\end{array}} \tag{49}$$

$$Exec-close_i\,\frac{true}{\begin{array}{r}<Exec-close_i,s>\rightarrow<Case-close_i,\\ s=params[Exec_i]>\end{array}} \tag{50}$$

$$Otherwise-open_i\,\frac{s.b=false}{<Otherwise-open_i,s>\rightarrow<Otherwise-value_i,s>} \tag{51}$$

$$Otherwise-value_i\,\frac{true}{\begin{array}{r}<Otherwise-open_i,s>\rightarrow<Otherwise-close_i,\\ params[Otherwise_i]=s.currentParam>\end{array}} \tag{52}$$

$$Otherwise-close\,\frac{true}{\begin{array}{r}<Otherwise-close_i,s>\rightarrow<Case-close_i,\\ s=params[Otherwise_i]>\end{array}} \tag{53}$$

$$Case-close\,\frac{true}{\begin{array}{r}<Case-close_i,s>\rightarrow<Case-Exit,\\ sParentStatus.currentParam=s>\end{array}} \tag{54}$$

Figure 15: Small-Step case statement specifications

fundamental executional constructs of the language. The same approach can be applied to all the CMML tags in the language.

7 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 main contribution of this paper is presenting a formal specifications for the CMML language through a Structural Operations Semantics SOS approach based on the Big-Step and the Small-Step methodologies. Moreover, a prototype of the overall framework was built and tested through a factorial ANOVA/GLM experiments and presented in [25].

Our future work will be concentrating on designing a virtual bare metal deployment mechanism that utilizes virtual resources for CMML metering engines deployment. The new approach aims at reducing the waste of resources used in the framework deployment targeting higher Return on Investment ROI, 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". *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pp. 285–292, June 2015.

[3] Krzysztof R Apt. "Ten Years of Hoare's Logic: A Survey – Part I". *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):431–483, 1981.

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

[5] 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.

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

[7] 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.

[8] D. Huemer and A.M. Tjoa. "A Stepwise Approach Towards an Interoperable and Flexible Logging Principle

for Audit Trails". *2010 Seventh International Conference on Information Technology: New Generations (ITNG)* , pp. 114 –119, April 2010.

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

[10] 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, pp. 276–279, New York, NY, USA, 2010. ACM.

[11] Gilles Kahn. *Natural Semantics*. Springer, 1987.

[12] 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, pp. 39–50, New York, NY, USA, 2010. ACM.

[13] Peter D Mosses. "Modular Structural Operational Semantics". 2005.

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

[15] 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". *2013 IEEE 12th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 26–33, June 2013.

[16] 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". *2013 22nd International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, July 2013.

[17] 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, Washington, DC, USA, 2007. IEEE Computer Society, pp. 93–101.

[18] GD Plotkin. *A Structural Approach to Operational Semantics*. 1981.

[19] Gordon D Plotkin. "The Origins of Structural Operational Semantics". *The Journal of Logic and Algebraic Programming* , 6061:3 – 15, 2004.

[20] 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-Aug. 2010.

[21] Michael Lee Scott. *Programming Panguage Pragmatics*. Morgan Kaufmann, 2000.

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

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

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

[25] Karim Sobh and Amr El-Kadi. "A Unified Cloud Metering Framework". *International Journal of Computers and Their Applications*, pp. 124 –139, 23(2), 2016.

[26] W3C. "Overview of GDML Resources". `https://www.w3.org/MarkUp/SGML/`, 2016. [Online; accessed 25-January-2016].

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

Karim Sobh is an Assistant Professor in the Department of Informatics and Computer Science at Nile University in Cairo. He 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. Sobh's 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. Moreover, he is the founder of 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

ISCA INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS Vol. 24, No. 3, Sept. 2017