Performance Aspects on Databases and Virtualized Real-time Applications

Christine Niyizamwiyitira

Blekinge Institute of Technology doctoral dissertation series
No 2018:02

Performance Aspects of Databases and Virtualized Real-time Applications

Christine Niyizamwiyitira

Doctoral Dissertation in Computer Systems Engineering

Department of Computer Science and Engineering
Blekinge Institute of Technology
SWEDEN

2018 Christine Niyizamwiyitira
Department of Computer Science and Engineering
Publisher: Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
ISBN: 978-91-7295-348-2
ISSN: 1653-2090
urn:nbn:se:bth-15758

Abstract

Context: High computing system performance depends on the interaction between software and hardware layers in modern computer systems. Two strong trends that effect different layers in computer systems are that single processors are now more or less completely replaced by multiprocessors, which are often organized into clusters, and virtualization of resources. The performance evaluation of different software on such physical and virtualized resources, is the focus of this thesis.

Objectives: The objectives of this thesis are to investigate

- The performance evaluation of SQL and NoSQL database, namely Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB;
- Soft real-time application namely, voice-driven web;
- Scheduling algorithms for resource allocation for hard real-time applications on virtual machine (VM).

Methods: Experiment is used to measure the performance of SQL and NoSQL databases, and to develop a prototype and predicts processor performance of voice-driven web on multiprocessors. Theoretical methods are used to model and design algorithms to schedule real-time applications on the VM. Simulation is used to quantify the performance of parameter values and to compare expected performance with theoretical bounds in schedulability tests.

Results: The performance of SQL and NoSQL database management systems namely, Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB is evaluated for writing and reading throughput and latencies. For reading throughput, all databases are horizontally scalable, however, only Cassandra and couchDB exhibit horizontal scalability for data writing. The overall evaluation shows that Cassandra has the most writing scalable throughput with a relative low latency, whereas PostgreSQL has the lowest writing latency, and MongoDB has the lowest reading latency.

The architectures' tradeoffs of voice-driven web show that the voice engine should be installed on the server side where its performance scales with the number of processors.

The thesis presents scheduling techniques for real-time applications in VM. VM's period and execution time that allow real-time applications to meet their deadlines are defined using these techniques. Simulation results show the impact of the length of different VM periods with respect to overhead, the tradeoffs between resources consumption and period length. A utilization based test for scheduling real-time application on virtual machine that determines if a taskset is schedulable or not is also presented. If the taskset is schedulable the algorithm provides the priority for each task. This algorithm avoids Dhall's effect that may cause tasksets with even very low utilization to miss deadlines.

Conclusions: The thesis presents the performance evaluation for reading and writing throughput and latencies for SQL and NoSQL databases. The thesis quantifies the tradeoffs of voice-driven web architectures and the performance scalability of the speech engine with respect to number of processors. Furthermore, this thesis proposes scheduling algorithms for real-time application with hard deadline on VMs.

Keywords: SQL and NoSQL database, Bigdata management systems, Voice-driven web, Multicore performance prediction, real-time Scheduling, Virtual Multiprocessor Scheduling.

Acknowledgements

I would like to thank my main supervisor, Professor Lars Lundberg. His guidance, generous and continuous support have been a great source of encouragement through the course towards the PhD degree. I am particularly grateful for his patience and humility in dealing with me. I would like also to thank my supervisor Dr. Mikael Svahnberg. I am particularly grateful for his inputs during the supervisory meetings, which have brought new perspectives and understanding during my PhD study. I would like to thank my senior reviewer Professor Håkan Lennerstad for his very good mathematical explanation as a part of my research. My time at Blekinge Tekniska Högskola (BTH) was made enjoyable in large part due to colleagues, friends and BTH staff that have become a part of my life. I would like to thank Sogand Shirinbab, Siva Krishna Dasari and Dr. Thi My Chinh Chu for many constructive discussions. Also, I would like to extend my acknowledgement to Eva-Lotta Runesson, Monica Nilsson, and the entire DIDD department for all the practical help they provided me. I would like to thank Dr. Louis Sibomana, Dr.Charles Kabiri and Dr. Said Ngoga Rutabayiro for supporting me socially and career wise both in Sweden and Rwanda. I am grateful to Anita Kayihura for enjoyable life in Sweden. I would like to thank Dr. Felix K. Akorli, for introducing me about BTH. I gratefully acknowledge the funding source that made my PhD studies possible, the University of Rwanda (UR) in partnership with the Swedish International Development Agency (SIDA). I thank the research project “Scalable resource-efficient systems for big data analytics” funded by the Knowledge Foundation (grant: 20140032) at BTH for giving me the opportunity to work on industrial problem through Telenor Sverige. Last but not least, I would like to express my deepest gratitude to my family, who have always been encouraging me throughout my studies.

Karlskrona, May 2018

Christine Niyizamwiyitira

Dedicated to Mucyo Faustin and Mucyo Isimbi Christa

List of publications

The thesis is based on the following publications:

1. C. Niyizamwiyitira and L. Lundberg, “Performance Evaluation of SQL and NoSQL Database Management Systems in a Cluster,” International Journal of Database Management Systems (IJDMS) Vol.9, No.6, December 2017

2. C. Niyizamwiyitira, L. Lundberg, and M. Svahnberg, “Evaluation of Voice-driven Web Application Architecture,” in Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on, 2012, pp. 555-562

3. C. Niyizamwiyitira and L. Lundberg, “Performance evaluation and prediction of open source speech engine on multicore processors,” in Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems, 2013, pp. 345-352

4. C. Niyizamwiyitira and L. Lundberg, “Real-time scheduling of multiple virtual machines,” International journal of Computers and their applications (IJCA) , Vol. 24, No. 3, pp.91-109, Sept. 2017

5. C. Niyizamwiyitira and L. Lundberg, “Period assignment in real-time scheduling of multiple virtual machines,” in Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems, 2015, pp. 180-187

6. C. Niyizamwiyitira, L. Lundberg, and H. Lennerstad, “A Utilization-based Schedulability Test of Real-time Systems Running on a Multiprocessor Virtual Machine,” accepted for publication in the Computer Journal

The following publications are associated with, but not included in this thesis:

1. C. Niyizamwiyitira, L. Lundberg, and H. Lennerstad, “Utilization-Based

Schedulability Test of Real-Time Systems on Virtual Multiprocessors," in Parallel Processing Workshops (ICPPW), 2015 44th International Conference on, pp. 267-276, Oct. 2015

2. C. Niyizamwiyitira and L. Lundberg, "Performance Evaluation of Trajectory Queries on Multiprocessor and Cluster," Computer Science & Information Technology (CS & IT), v.6, pp. 145-163, May 2016

Table of Contents

Chapter One

1.1 Motivation

Providing high computer system performance has been a key challenge since the design of the very first computers. Today, performance is an intricate function of the interaction between different software and hardware layers in modern computer systems. Two strong trends that affect different layers in modern computer systems are that single processors are now more or less completely replaced by multiprocessors, which are often organized into clusters, and that more and more resources are being virtualized.

The multiprocessor and cluster trend is mainly driven by hardware design issues, e.g., modern processor chips are designed with a number of cores. The trend with software virtualization of hardware resources such as processors, network and storage, is mainly driven by the need for high and cost-effective resource utilization.

One important class of computer system applications is so called real-time applications. A real-time application is an application for which the correctness of the system requires not only a correct logical output for a certain input, which is a fundamental correctness requirement in all computer systems; a correct real-time application (or real-time system) must also present the correct output within certain time limits, otherwise the output is of no value (in the case of so called hard real-time systems) or of significantly reduced value (in the case of so called soft real-time systems). This means that high and predictable performance is a key aspect in real-time systems. There are numerous examples of real-time systems, ranging from embedded controllers in washing machines, to the systems that control the breaks in our cars, to control systems in fighter aircrafts, to systems that control nuclear plants and huge telecom networks. The common aspect is that these systems need to handle and interact with real-world events within certain time limits. Such events must be handled in a correct and timely manner in order to avoid that the car, nuclear plant or plane will not crash, and to make sure that users will not suffer from frozen images etc. when streaming video or music over a telecommunication network.

One of the main research areas in real-time systems is how to schedule and prioritize different activities in a computer system so that one can guarantee that the system will respond within certain time frames. A real-time scheduling strategy should address two aspects: it should define a priority order for the different activities (or tasks), and it should provide a so called schedulability test which makes it possible to determine if the application will indeed meet all its deadlines using this priority order. Both multiprocessors and virtualization introduce new challenging when it comes to scheduling and prioritization of such activities. Finding techniques to handle these new challenges is very important so that real-time applications can benefit from these two important technologies.

Many applications, including hard and soft real-time applications, use database systems to handle the big data that are facing almost all applications. Understanding and being able to predict the performance of different database technologies is therefore very important.

In this thesis I investigate the performance of different database systems. I also investigate the performance of a soft real-time application called voice-driven web for different software architectures. In the context of real-time applications in virtualized environments, I have designed schedulability tests and algorithms for virtualized single- and multiprocessor systems.

1.2 Historical Background

The background of this thesis starts with a brief history of microprocessor. A microprocessor is the heart of Central Processing Unit (CPU), CPU is the controlling element of a computer system. It all started in early 1970s when Intel developed its first microprocessor, a 4 bit microprocessor 4004. This microprocessor was followed by the 8 bit microprocessor Intel 8008 and Intel 8080 in 1971 and 1973 respectively. About the same time, Motorola released its first microprocessor, an 8 bit 6800 which had approximately the same processing power as Intel 8080. Based on 6800, Motorola developed other microprocessors namely, 6805, 6808, and 6811. Motorola processors have kept evolving until the PowerPC version. Meanwhile Intel improved to 16 bit microprocessors, 8086, and 8088.

Intel 8088 is the first microprocessor that was implemented in personal computers (PC) by IBM in 1981. Intel 8086/8088 served as the basis for Intel 80386, a 32 bit microprocessor that was developed in 1986. Intel kept improving their microprocessors until the Pentium microprocessor was introduced in 1993. From generation to generation, microprocessor grew smaller and faster, with heat dissipation and more energy consumption. Intel Pentium evolved into Pentium 1, Pentium 2, Pentium3, and Pentium 4 versions that have higher clock respectively. Those are, Pentium 2, Pentium 3, Pentium 4, and Core2 which marked the end of the evolution of the 32 bit microprocessor architecture. Pentium 4 was first introduced in 2000 [1].

As PCs became pervasive, many applications have been designed to run on them. Later, these applications required more and faster processing because of the growing amount of data that they need to handle. The initial response to this demand was to increase a processor speed by boosting its clock speed. In 1975, Gordon Moore stated that the number of transistors doubles every two years for increasing the computer performance[2], [3]. However, increasing the clock speed increases the power consumption and the heat dissipation to extremely high levels, thus making the processor impractical. Due to advances in circuit integration technology and performance limitations in wide-issue, super speculative processors, Chip-Multiprocessor (CMP) or multi-core technology has become the mainstream in CPU designs [4]. The above CPU design are supported by superscalar and VLIW (Very Long Instruction Word) design. Super-scalar and VLIW implement a form of parallelism called instruction level, i.e., dispatch multiple instructions during a single clock cycle. Super-scalar issue instruction dynamically whereas VLIW issues instructions statically. Besides that, Simultaneous multithreading (SMT) improves the overall efficiency of superscalar and VLIW CPU through hardware multithreading (multithreading is called hyperthreading in Intel processor) [5].

The Core2 microprocessor was introduced in 2006. It had an architecture that supports more than one core on a die [1]. The current microprocessors, such as dual core, quad core etc. that we have in our computers, are microprocessor generations of the Core2 microprocessor family. In 2008, there existed two and four cores processors in 2009, AMD releases quad core processors Phenom and Athlon while Intel releases Core 2 Quad processor Q8400. In 2010, AMD releases hex/six core processor while Intel releases Core 2 Quad processor, Core i5, Core i3 Desktop processors and mobile processors. In 2011, AMD releases mobile and desktop processors in A4, A6, and A8 line, these processors support bigger cache memory respectively. In 2011 Intel releases Core i5 processors with 4 cores. In 2012 AMD releases desktop processors in A10 line, Intel releases Core i7 processor that is almost as fast as Core i3 and core i5 but with addition hyper threading feature that let each core to be used twice which makes the impression of doubling the number of cores. In 2017 Intel releases core i9 processors. As technology evolves, cluster computing and virtualization are two trends that increase the performance of processors.

1.2.1 Cluster Computing

During the past decade many different computer systems that support high performance computing have emerged. Their taxonomy is based on how their processors, memory, and interconnection are organized. Among such systems, there are so called clusters. In a cluster, different computers work together in order to achieve the same goal. Nodes (computers) are interconnected in a local area network (LAN), these nodes have the same type of hardware and the same operating system. A node can be a single or a multiprocessor system (PCs, workstations, or symmetric multiprocessor systems) with memory, I/O facilities, and an operating system. A cluster generally refers to two or more nodes connected together [6]. Applications that require high performance computing benefit from cluster. In this thesis, the performance of database systems, is evaluated on a cluster based environment.

1.2.2 Virtualization

The concept of virtualization has its origins in the mainframe days in the late 1960s and early 1970s, when IBM invested a lot of time and effort in developing robust time-sharing solutions. Time-sharing refers to the shared usage of computer resources among a large group of users, aiming to increase the efficiency of the expensive computer resources the users share. This model represented a major breakthrough in computer technology. Similar reasons are driving virtualization for industry standard computing today: the capacity in a single server is so large that many workloads cannot use it effectively. The best way to improve resource utilization, and at the same time simplify data center management, is through virtualization. Data centers today use virtualization techniques to make abstractions of the physical hardware, create large aggregated pools of logical resources consisting of CPUs, memory, disks, file storage, applications, networking, and offer those resources to users or customers in the form of agile, scalable, consolidated virtual machines. Even though the technology and use cases have evolved, the core meaning of virtualization remains the same: to enable a computing environment to run multiple independent systems at the same time [7]. Virtualization becomes more important in cloud computing whereby it helps to deliver shared resources as service on demand.

In order to monitor and control different users that are time sharing the hardware used for virtualization, a layer called hypervisor between the users and the hardware creates a virtual platform on the host hardware. On top of the virtual platform multiple guest operating systems are executed and monitored. In this way, multiple operating systems, which are either multiple instances of the same operating system, or different operating systems, can share the hardware resources offered by the host. Hypervisors are classified as native (or bare metal), and as hosted. Native hypervisors are software systems that run directly on the host's hardware to control the hardware, and monitor the guest operating systems. Consequently, the guest operating system runs on a separate level above the hypervisor. Example of those are VMware and Xen [8], [9]. Hosted hypervisors are designed to run on top of a traditional operating system. In other words, a hosted hypervisor adds a distinct software layer on top of the host operating system, and the guest operating system becomes a third software level above the hardware. Example of those are KVM and Virtual box[10], [11]. One of the important applications nowadays is real-time application, this thesis proposes the schedulability algorithms of real-time application in virtual environments.

1.2.3 Database Applications

A database application is defined as a collection of information that is organized in order to be easily retrieved, managed, and updated. A Database Management System (DBMS) is a collection of interrelated data and a set of programs to access the data. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient [12].

From the earliest days of computers, storing and manipulating data has been a major focus. The first general-purpose DBMS was designed by Charles Bachman at General Electric in the early 1960s and was called the Integrated Data Store. It formed the basis for the network data model, which was standardized by the Conference on Data Systems Languages (CODASYL) and strongly influenced database systems through the 1960s. In 1970, Edgar Codd, at IBM's San Jose Research Laboratory, proposed a new data representation framework called the relational data model. Relational DBMS (RDBMS) have become the most popular DBMS[13]. According to database models [14], the most known models are given below;

- *Relational Database* is a database that consists of a collection of tables, each of which is assigned a unique name. A table is a collection of records, and each record is a collection of fields (attribute values). In a table, attributes are represented as columns whereas records are represented as rows. In general a row in a table represents a relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of table and the mathematical concept of relation, from which the relational data model takes its name. Examples of relational database are MySQL, Oracle, Microsoft SQL.

- *Network Database* is a database that consists of a collection of records connected to one another through links. A record is in many respects similar to an entity in the entity relationship (E-R) model. Each record is a collection of fields (attributes), each of which contains only one data value. A link is an association between precisely two records. Thus, a link can be viewed as a restricted (binary) form of relationship in the sense of the E-R model [15]. An example of a network database is Integrated Database Management System (IDMS).

- *Hierarchical Database* is a database that consists of a collection of records that are connected to each other through links. A record is similar to a record in the network model. Each record is a collection of fields (attributes), each of which contains only one data value. A link is an association between precisely two records. Thus, a link here is similar to a link in the network model. The schema for a hierarchical database consists of boxes which correspond to record types and lines which correspond to links [15]. An example of hierarchical database is IBM's Information Management System (IMS).
Record types are organized in the form of a rooted tree and no cycles in the underlying graph. Relationships formed in the graph must be such that only one-to-many or one-to-one relationships exist between a parent and a child. A parent may have an arrow pointing to a child, but a child must have an arrow pointing to its parent.

- *Object Oriented Database* is a database that has been proposed as an alternative to relational systems and such systems are aimed at application domains where complex objects play a central role. The approach is heavily influenced by object-oriented programming languages and can be understood as an attempt to add DBMS functionality to a programming language environment such as Python, Java, and Objective-C [15]. Example of an object oriented database is ObjectDB.

- *Relational Object Oriented Database* can be thought of as an attempt to extend relational database systems with the functionality necessary to support a broader class of applications and, in many ways, provide a bridge between the relational and object-oriented paradigms. Relation object database extends the relational data model by including objectorientation and constructs to deal with added *data types*. It allows attributes of *tuples* to have complex types, including non-atomic values such as nested relations, e.g., an address with subpart like street, post number, etc. Relational object oriented database systems preserve relational foundations, in particular declarative access to data, while extending modelling power. It again upwards compatibility with existing relational languages [14], [15]. Example of a relational object oriented database is PostgreSQL.

- *Column Oriented Database* is a database that stores data tables as sections of columns of data rather than as rows of data, i.e., it stores all values of the same attribute of the relational conceptual schema relation together [16]. In comparison, most relational DBMSs store data in rows, it is more beneficial to use row-oriented storage structure if there is mostly transaction queries performed on a database. Those transactions are known as OnLine Transaction Processing (OLTP). OLTP queries imply a set of reads and writes to a few rows at a time. However, a column-oriented storage structure is more beneficial if there are mostly analytical queries performed on a database; those analytical queries are known as OnLine Analytical Processing (OLAP). OLAP queries imply bulk updates and large scans of a few columns but many rows (e.g., aggregate values calculation). Columns compress better than rows, typically row-store compression ratio is 1:3 whereas for column-store is 1:10. The reason is that rows contain values from different domains, hence more entropy, difficult to dense-pack. Columns exhibit significantly less entropy [17], [18]. An example of column oriented database is Cassandra.

- *Document Oriented Database* is a database that stores data as XML files. The advantage is that the XML format is widely accepted, a wide variety of tools are available to assist in its processing and XML has become the dominant format for data exchange over internet. Compared to storage of data in a relational database, the XML representation may be inefficient, since tag names are repeated throughout the document. However, in spite of this disadvantage, an XML representation has significant advantages when it is used to exchange data between organizations, and for storing complex structured information in files [15]. An example of a document oriented database is MongoDB.

- *Graph Database* is a graph, which is just a collection of vertices and *edges*. In other words, a set of nodes and the relationships that connect them. For example, Twitter’s data is easily represented as a graph, the relationship between people is indicated by the edge “follows”, i.e., on Twitter or Facebook, users can see who follows who, who likes what,…, example of a graph database is Neo4J, Neo4J uses an expressive querying language called Cypher [19].

- *Flat file Database*, a flat file database describes any of various means to encode a database model (most commonly a table) as a single file. This table has rows and columns with the same meaning as of relational database. There are usually no structural relationships between the records. One limitation of flat file databases is data redundancy [20]. Example of a flat file database is Microsoft Excel.

- There exist some database systems that combine two models, e.g., column oriented and relation database such as SAP HANA. SAP HANA is an in-memory, column oriented relational database system management developed by SAP SE in 2010 [21]. SAP partners with Hasso Plattner Institute (HPI) to improve SAP HANA [22]. The goal of the SAP HANA database is the integration of transactional and analytical workload within the same database management system. Since SAP HANA is an in-memory database, it relies on the main memory for computer storage in contrary to disk storage mechanism that is used but mostly other database management systems. Main memory database systems are faster than disk-optimized database systems because the disk access is slower than memory access, however, main memory is more expensive than disk storage, and therefore a drawback could be the size of the main memory with respect to big data processing requirements. The architecture of SAP HANA allows online transaction processing (OLAP) [23], [24].

- There are two main revolutions in data management, namely Big Data analytics and not-only Structured Query Language (NoSQL), aiming at making real-time decisions using volume of complex data sets that could be both structured and unstructured [25]. An example of a structured table is given in Table 1. All the rows (records) have the same attributes (same number of columns) and there is a primary key. An unstructured table (see Table 2) does not necessary need to have the same number of attributes. Every record has an automatic unique identification that is generated by the database system (see the first column of Table 2).

Table 1. Structured Table

ID (primary key)	**Name**	**State**	**Birth-Date**
12	Bill	DC	1985
25	Howard	PA	1984
46	David	NY	1956
11	Edward	CA	1976

Table 2. Unstructured Table

7b976c48	Name: Bill	State: DC	Birth-date:1985
7c8f33e2	Name: Howard	State: PA	Birth-date: 1984
7d2a3630	Name: David	State: NY	
7da30d76	Name: Edward	State: CA	

- The presence of unstructured data stimulated the invention of new database systems since Relational Database Management Systems

(RDBMS) uses Structured Query Language (SQL) to deal with structured data only. Relational database has been the default choice for data model adoption in businesses worldwide over the past thirty years with Structured Query Language (SQL). However with the rapid increase of data, SQL has become inefficient. As a result, NoSQL was introduced with a set of new database management features, such as flexibility towards data structure and horizontal scalability for big data processing [26], [27].

The main features of NoSQL follow the CAP (Consistency, Availability, and Partition tolerance) theorem [28]. The core idea of CAP is that a distributed system cannot meet the three distinct needs, i.e., Consistency, Availability, and Partition tolerance, simultaneously. According to data storage models, NoSQL can be, key value based, column oriented based, document oriented based, and graph oriented based. Column oriented based, document oriented based, and graph oriented based are explained previously in section 1.2.3.

Key value data model means that a value corresponds to a key, i.e., a string that represents the key and the actual data that represents the value, thus creating a “key-value” pair. The performance aspects of a database system is an important issue, since very large data repositories are increasing rapidly. Some of the most popular SQL and NoSQL open source database systems are evaluated in this thesis, those are Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB.

- *Cassandra* is an open-source NoSQL column based database that is written in Java. It is a top level Apache project born at Facebook and built on Amazon’s Dynamo and Google’s BigTable. It is a distributed database for managing large amounts of structured data across many commodity servers, while providing highly available service and no single point of failure. In CAP, Cassandra has availability and partition tolerance (AP) with eventual consistency. Cassandra offers continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones. Cassandra has a masterless ring architecture [29]. Keyspace (Cassandra concept that mean a collection of tables) is similar to database in RDBMS, inside keyspace there are tables which are similar to tables in RDBMS, column and rows are similar to those of RDBMS' tables. The querying language is Cassandra Query Language (CQL) that is similar to SQL in RDBMS. Cassandra does not natively support spatial indexing but this can be extended via Stratio's Cassandra Lucene index. Stratio's Cassandra Lucene Index is a plugin for Apache Cassandra , it extends Cassandra's index functionality to provide near real time search [30].

- *CouchDB* is a database that is written in Erlang, it stores data with JSON documents. CouchDB accesses documents and queries indexes with a web browser, via HTTP. Indexing, and transforming documents are perfomed using JavaScript. CouchDB is highly available and partition tolerant, but is also eventually consistent, CouchDB supports masterless setup [31]. CouchDB does not natively support spatial queries, GeoCouch extends CouchDB's spatial indexing [32].

- *MongoDB* is an open-source NoSQL document database; MongoDB is written in C++. MongoDB consists of a database, that contains collections, these are like tables in RDBMS, inside a collection there are documents, these are like a tuple/row in RDBMS, and inside a document there are fields which are like column in RDBMS [33], [34]. MongoDB is consistent and partition tolerant. MongoDB has natively a built in function for spatial queries and it has a sharding feature to support horizontal scalability of the database ina master/slave fashion [35].

- *PostgreSQL* is an open source object RDBMS that is written in C, it has two features according to the CAP theorem. Those are availability, i.e., each user can always read and write and consistency, i.e., all users have the same view of data. PostgreSQL organises data in columns and rows [36, p. 3]. PostgreSQL does not natively support horizontal scalability and spatial queries, PostgreSQL is extended by CITUS and

PostGIS to support scalability in a master/slave fashion and spatial queries indexing respectively [37], [38].

- *RethinkDB* is an open source NoSQL database that is written in C++.Rethink is horizontally scalable in a master/slave fashion, it is mostly designed to facilitate real-time updates for query results to applications [39]. RethinkDB natively supports spatial queries using GeoJson. The database uses the ReQL query language that is available for Python, Ruby, and Java.

1.2.4 Voice-driven Web Applications

Voice-driven web is a Voice XML soft real-time application that provides automated interaction for callers, making it possible to retrieve information from the web through a telephone keypad or speech recognition. It controls and responds to callers by using speech technologies [40]. The architecture of voice-driven web has three main parts; (1) a voice browser that interacts directly with the terminal, (2) a speech technology server, i.e., Automatic Speech Recognition (ASR) that recognizes the input speech from the end-user and Text To Speech (TTS) that reads the content from the web in response to the end-user request, and (3) the back end which is composed of a web proxy and the World Wide Web (www). Figure 1 shows the voice-driven web architecture. ASR and TTS are the most computationally demanding parts in the voice-driven web architecture. Hence, implementing these parts on powerful processors would improve the overall response time which is an important factor for the quality of voice-driven web applications.

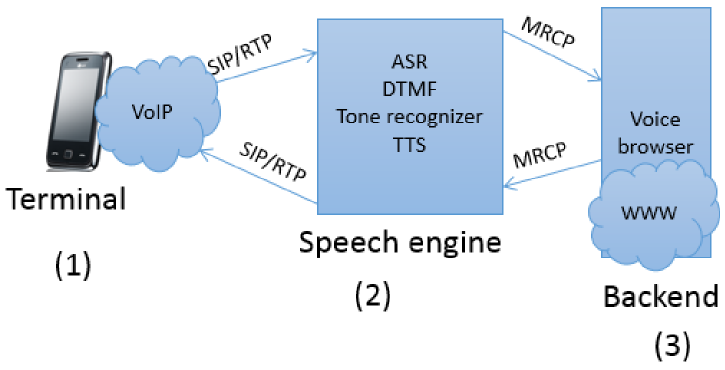


Figure 1. Architecture of voice-driven web.

1.2.5 Real-time Applications

Real-time applications are those applications for which the time that the output is produced matters very much. It can also be defined as an application that has to respond to a stimuli within a finite and specific period. This definition covers a very wide range of computer activities, e.g., an operating system like windows, Linux, maybe considered to be real- time in that when a user enters a command the response must be prompted within seconds. Fortunately, it is not a disaster if the response is not forthcoming. These real-time applications are distinguished from those when a system fails to responds at a correct time, thus the response is wrong. Therefore, real-time application are distinguished by the correctness of the response which depends not only on the logical results but also on the time at which results are produced. Consequently, we classify real-time application as hard or soft on the basis of severity of meeting its deadlines. Hard real-time applications are those that the response must absolutely occur within a specific and explicit deadline, e.g., flight controller, if a deadline is missed it could lead to catastrophe, a car brake, etc. For soft real-time applications, the system will still function if deadlines are occasionally missed, e.g., video and audio streaming, and voice driven web applications.

Example of a real-time application with 3 tasks $\{\tau_1, \tau_2, \tau_3\}$, every task has a period *T* and execution time *C*. A task with shorter period has higher priority using the common scheduling algorithm Rate Monotonic (RM). In figure 2, it is clear task τ_1 has the highest priority.

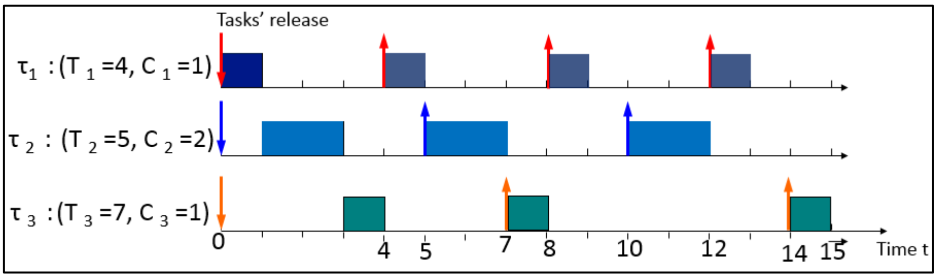

Tasks' release
τ_1 :(T_1=4, C_1=1)
τ_2 : (T_2=5, C_2=2)
τ_3 :(T_3=7, C_3=1)
0
4
5
7
8
10
12
14
15
Time t

Figure 2. Real-time tasks

Real-time application's tasks are classified as follows [41];

- *Periodic tasks*: a periodic task has a regular inter arrival time equal to its period, and a deadline that is shorter than its period, or at most the deadline coincides with the end of its current period. Periodic tasks usually have hard deadlines, but in some applications the deadlines can be soft.
- *Aperiodic tasks:* aperiodic tasks arrive at irregular intervals, i.e., tasks respond to randomly arriving events. The minimum separation between two consecutive tasks can be zero if more tasks of an aperiodic real-time application occur at the same time instant. Aperiodic tasks typically require a fast response time.
- *Sporadic tasks:* These are aperiodic tasks with a minimum inter arrival time. Note that without a minimum inter arrival time restriction, it is impossible to guarantee that a deadline of a sporadic task would always be met.

In this thesis, the performance of virtual processors is evaluated when handling critical systems that require strict regular resource and are time constrained. Those are real-time applications with hard deadline. Real-time applications with hard deadlines are important applications that need to be predicted if they are schedulable on a given resource in advance in order to meet their deadlines. Moving a real-time application 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 is considered in this thesis, where a number of VMs share the same processor core, and each VM contains a real-time application.

Moreover, for processors that have many cores, virtualization makes it possible to run virtual multiprocessor machine on one physical multiprocessor, and it is possible to configure that VM to use all or a subset of the physical cores. Different techniques and algorithms are proposed to schedule the processor resources so that real-time applications meet their deadlines. A real-time application is a sequence of real-time tasks that execute according to priority. Real-time applications are classified as fixed or dynamic priority. In fixed priority real-time applications, every task has a priority that does not change in all periods. In dynamic priority based real-time applications, the priority of a task depends on the current situation, this may change the task priority in different periods.

Scheduling real-time applications on virtual machines happens at two levels [42], [43]. The first level is traditional real-time scheduling of the tasks within a VM. The second level is scheduling of VMs by the hypervisor on the 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 [44]. These kinds of scheduling algorithms make it 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, given the blocking time Bp that a VM does not have access to the processor, it is necessary to determine a period T_{VM} and a (worst-case) execution time C_{VM} for the VM such that all real-time tasks in the VMs will meet their deadlines.

1.3 Objectives and Research Questions

The objective of this thesis is to investigate the performance evaluation of SQL and NoSQL database systems, namely Cassandra, CouchDB, MongoDB, PostgreSQL, RethinkDB; and a soft real-time application, namely voice-driven web. Furthermore, schedulability algorithms for resource allocation for hard real-time applications on virtual processors, are proposed. In order to achieve the thesis' objectives, the four research questions that guide this thesis are,

Research Question One: How do SQL and NoSQL database systems perform on clusters?

Research Question Two: How does a voice engine's performance scale with respect to increasing of number of processors in a multiprocessor?

Research Question Three: How to schedule the processor resources when a number of VMs share a single core processor, so that real-time applications meet their deadlines?

Research Question Four: Based on the utilization, how to test the schedulability of real-time application on a virtual multiprocessor machine?

1.4 Research Methodology

Three different methods are used to conduct this thesis; those are experiment, theoretical research, and simulation.

Experiment is a research method that helps to predicts and measure phenomena about a thing. Experiment is used at different stages, such as development, evaluation, and problem solving research. Experiment is performed under a controlled environment. [45].

Theoretical research is a way of developing basic theory, mathematical models, and proofs. This research method can be appropriate for formal sciences, e.g., mathematics and parts of computer science [46], [47].

Simulation is a way of imitating the real world and model it on a computer with the purpose of prediction, performing some tasks, training, proof, and theory discovery [48].

In Chapter two, the experiment method is used to evaluate the performance of database systems in cluster computing.

In Chapter three, experiments are used to evaluate different architectures of voice-driven web and in this chapter prototypes are developed in the same chapter I also evaluate the performance of a voice engine on a multiprocessor.

Theoretical research is used in Chapter four and five with mathematical modeling which are used to design algorithm and techniques that are used for processor resource allocation in virtual machines so that real-time applications that run on top of those machines meet their deadlines. Simulations are used to quantify the performance implications of certain parameter values in our theoretical results in Chapter four. Simulations are also used to compare expected performance with theoretical bounds in our schedulability tests in Chapter five. Matlab is used to conduct our simulations.

1.5 Thesis Overview

In order to understand multicore processor and multiprocessor cluster performance towards different applications, this thesis reports performance evaluation of SQL and NoSQL database systems, notably Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB; and a soft real-time application called voice-driven web. Virtualization gives the ability to run multiple operating systems on a single physical hardware, whereby the operating systems share underlying physical hardware resources. In order to benefit from virtualization technology, one would like to move real-time applications to virtualized processors. In order to guarantee that a real-time application is schedulable on a given virtual processor, scheduling algorithms must be developed. This thesis proposes scheduling algorithms for hard real-time application on virtual processors.

Figure 3 depicts the overview of the thesis. I investigate the performance of SQL and NoSQL database systems on clusters. I also present the architecture and quantify the performance of a voice-driven web application on a multiprocessor. I finally provide schedulability algorithms of hard real-time applications on virtual processors. Figure 4 shows the structure of the thesis.

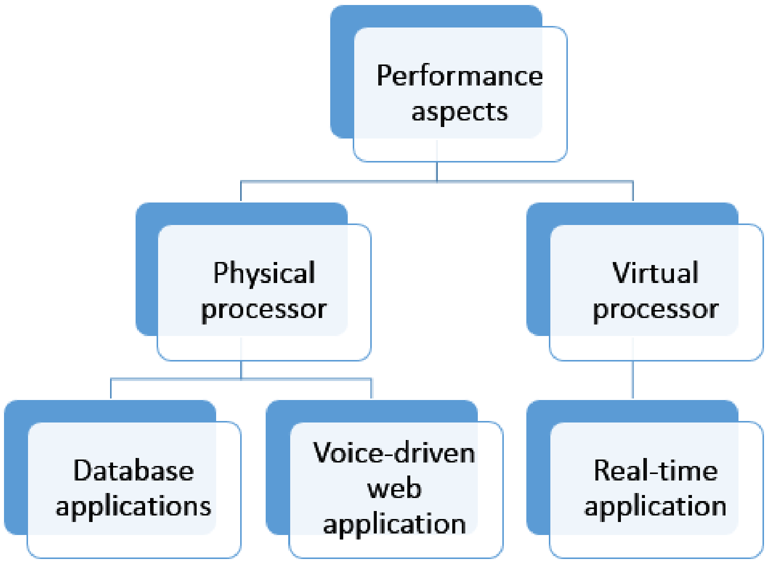
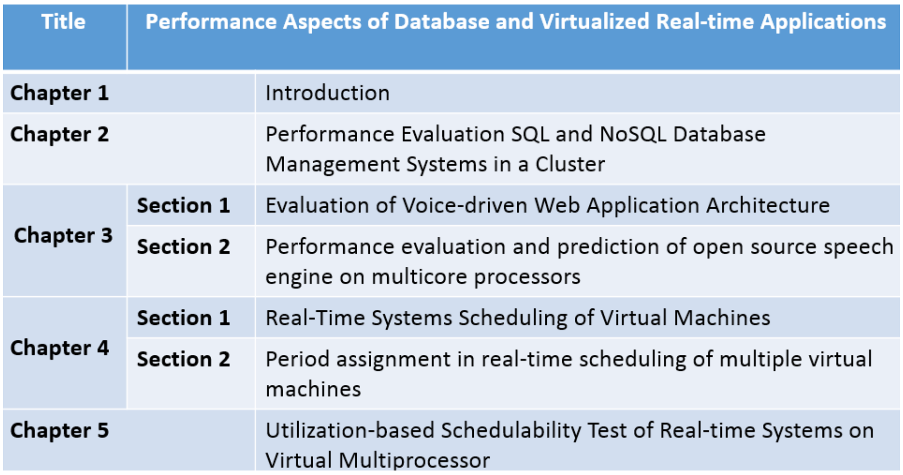


Figure 3. Thesis Overview



Title		Performance Aspects of Database and Virtualized Real-time Applications
Chapter 1		Introduction
Chapter 2		Performance Evaluation SQL and NoSQL Database Management Systems in a Cluster
Chapter 3	Section 1	Evaluation of Voice-driven Web Application Architecture
	Section 2	Performance evaluation and prediction of open source speech engine on multicore processors
Chapter 4	Section 1	Real-Time Systems Scheduling of Virtual Machines
	Section 2	Period assignment in real-time scheduling of multiple virtual machines
Chapter 5		Utilization-based Schedulability Test of Real-time Systems on Virtual Multiprocessor

Figure 4. Thesis Structure

1.6 Thesis Contributions

This thesis presents performance of SQL and NoSQL database management systems, Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB as well as a soft real-time application called voice-driven web. This thesis also presents schedulability algorithms for hard real-time applications on virtual processors. The results are useful for:

- Application developers: to improve software.
- Research and education community: serve as the base for new research findings.
- Business: business decision support such as selecting a proper computing environment, sharing resource in Infrastructure as Service (IaaS) in cloud computing.

The main contributions of this thesis are,

Chapter Two: This chapter focuses on the performance of SQL and NoSQL database management systems namely Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB executing on a cluster. The evaluation shows how fast these database systems process queries according to the number of computer nodes in the cluster. The results help in decision support process in order to choose proper hardware that fits application requirements.

Chapter Three: A voice-driven web architecture is investigated and prototypes are implemented. The tradeoffs between different architectures are presented. The architecture that implements speech engine Automatic Speech Recognition and Text To Speech on the server side is found to be scalable with the number of processors. The performance of this speech engine is quantified on a multicore processor with and without hyperthreading. The response time and the speedup are presented with respect to number of cores in the processor.

Chapter Four: This chapter presents scheduling techniques for real-time applications on virtual machines (VM). Techniques for virtual processor resource allocation for periodic real-time applications presented in this thesis makes possible to reuse classical scheduling techniques such Rate Monotonic Scheduling on the hypervisor level where the hypervisor maps VMs on the physical processors. A virtual machine requires a period and execution time in order to run real-time applications, this chapter explains how to determine the VM's period and execution time with or without overhead. Simulation shows the impact of the length of different VMs' periods whether the VMs' overhead is present or not. The tradeoffs between resources consumption and period length are also detailed.

Chapter Five: This chapter proposes an algorithm to schedule real-time tasks on a virtual multiprocessor machine. This algorithm provides a utilization-based test that shows if a task set is schedulable or not. If the task set is schedulable the algorithm provides the priority for each task. This algorithm avoids Dhall's effect, which may cause task sets with even very low utilization to miss deadlines.

1.6 Related Work

Performance aspects of database systems and voice-driven web have been investigated. Different techniques for scheduling virtual processors resource for real-time applications with hard deadline have been studied.

Insights on how to improve applications in order to benefit from the scalability of processors have been also revealed for some specific applications. Furthermore, results of the evaluation are recommendations to make a proper choice of hardware platform given the application's requirements. The following literature provides previous related work.

Query processing on multiprocessors has been studied in [49]. The authors implemented an emulator of a parallel DBMS that uses cluster that consists of multiprocessor nodes. In this thesis, query processing is evaluated on real physical hardware with existing general purpose databases (Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB). Query processing on FPGA and GPU on spatial-temporal data was studied in [50]. The authors present a FPGA and GPU implementation that process complex queries in parallel. However the study did not investigate the performance of various existing database systems. Also, the distributed environment was not also evaluated, whereas, in this thesis query processing are investigated on various SQL and NoSQL database systems running on a cluster. In [51], the authors conducted a survey on mining massive-scale spatio-temporal trajectory data based on parallel computing platforms such as GPU and FPGA, in this thesis existing general purpose databases are evaluated.

In [52], the authors presented an architecture for voice-enabled interfaces over local wireless networks, they found that the use of speech synthesis is preferred over the use of pre-recorded prompts. This increases flexibility of the dialogue and an easier upgrade of the applications. Due to wireless networks fluctuation, synthesized speech would have good quality once implemented on the mobile device. However, due to ergonomic and economic reasons, the embedded CPU and memory resources are very limited in the mobile device. Since speech synthesis systems require a lot of CPU, it is proposed to be implemented on a separate server. In this thesis, voice-driven web architectures are evaluated based on software architecture qualities, the tradeoffs between those architectures are also quantified.

The performance of FreeTTS that is implemented in Java, and C was evaluated on a dual core processor in [53]. The experimental results showed that FreeTTS that is implemented in Java performs better in terms of response time than the one implemented in C. These authors also evaluated the performance of Sphinx-4 (Automatic Speech Recognition) and they presented results on a dual core processor [54]. In this thesis, FreeTTS and Sphinx-4 are studied on a higher number of processor cores, up to eight cores and that hyperthreading is also considered.

In [55], the authors 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 considers each VM in isolation, i.e., without considering other VMs that could share the processor. In this thesis, an improved VM execution time C_{VM_i} and period T_{VM_i} are defined by considering a holistic perspective. It means that the whole work-load of all VMs that share a processor core is considered. As a result, more real-time concurrent applications become schedulable on a single core processor. The presence of overhead for context switches between VMs is also considered. If the overhead is not considered, it means that there are no co-located VM, therefore the hypervisor offers resource assuming that virtual machines are isolated. As results, this could lead to excessive use of CPU resources (referred as overprovisioning in datacenter) and low network throughput due to neglected inter-VMs communication overhead [56], [57]. In this thesis, period assignment techniques that compensate the overhead from co-residents VMs are presented.

There are two approaches to schedule real-time applications on multiprocessor systems, partitioned and global. In the partitioned approach, a task is statically allocated to a particular processor core, i.e., a task is not allowed to migrate to another processor. In the global approach, any tasks can execute on any processor core. The theory behind uniprocessor scheduling can be used for partitioned multiprocessor scheduling, e.g., the well-known uniprocessor schedulability test [44]. However, when it comes to global scheduling, the so called Dhall’s effect shows that a direct application of the RM priority assignment scheme may result in failure to meet deadlines even for systems with very low multiprocessor utilization regardless of how many processors are used [58]. In [59], the authors presented a method to schedule real-time applications on multiprocessors by virtualizing the processors. Tasks are clustered and each cluster is assigned to a set of virtual processors. This mean that the scheduling happens at two levels. The first is scheduling virtual processors onto physical processors and the second is scheduling tasks on a set of virtual processors.

In virtualized systems, it is not possible to define a schedulability bound based on a fixed (multiprocessor) utilization. The reason for this is that even real-time application with very low utilization may miss deadlines if the period of the task is shorter than the length of the time interval that the virtual processor does not have access to the physical hardware. In [60], the authors studied a scheduling method for real-time applications on multicore processors using a concept called synchronized deferrable server. They categorize tasks into non-migrating tasks and migrating. Non-migrating tasks are statically bound to a core. The schedulability analysis of non-migrating tasks is essentially a uniprocessor scheduling using existing scheduling algorithm such as RM. Migrating tasks are modelled as sporadic tasks and are allowed to migrate across the cores and thus can be processed by deferrable server. This results in using partitioning and global scheduling at the same time which is an advantage since neither partitioning scheduling nor global scheduling outperforms the other in all cases [61]. In this thesis, an algorithm that provides a utilization based test is proposed. The test classifies tasks into two priority classes, namely heavy and light tasks. Heavy tasks have higher priority than light tasks. The algorithm checks each task in the task set in order to make sure that the whole task set is schedulable.

1.7 Research Validity

In order to accurately measure the performance of SQL and NoSQL management systems on a cluster. New servers are used, i.e., apart from operating system, these database systems are the first applications that are installed and are the only applications that are running on those servers during the time of experiment. Stable releases of open source databases are considered and are used in order to allow reproducibility.

The results of voice-driven web architecture are based on measurements that have been recommended and used by a wide range of researchers.

Prototypes use real-world data such as live weather forecast data. Recordings are done from a native English speaker, and prototypes can be tested using telephone calling from any types of telephone. In order to evaluate the performance of voice-driven web speech engine on variety of processors, measurement are conducted on two different types of processors. Intel core i 7 quad-core and Intel-xeon dual quad-core. In order to explore processor features, measurements are conducted on those processors with and without hyperthreading. In order to allow reproducibility, open source Automatic Speech Recognition and Text To Speech (Sphinx-4 and FreeTTS) are used.

For real-time applications schedulability in virtual machines, algorithms are proposed and they are proved using mathematical formulation. These algorithms are supported by simulation studies that apply algorithms on a large number of tasks in a real-time application. A limitation is to test these algorithms in real-world environment like in cloud at a layer of Infrastructure as Service, where resource could be shared among real-time applications using the proposed algorithms. In order to cope with this, the simulation includes an overhead model that mimics the real-world infrastructure behavior.

1.8 Conclusions

The main purpose of this thesis is to investigate the performance of SQL and NoSQL database management systems and real-time applications, and propose schedulability algorithms for real-time applications on virtual processors. This thesis is guided by four research questions.

Research question one is the performance of SQL and NoSQL management systems on a cluster; this question is addressed in Chapter two. The performance of open source database systems, namely Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB are evaluated for different queries on a cluster. The results show that Cassandra has the most writing scalable throughput with relative low latency, whereas, PostgreSQL has the lowest writing latency. MongoDB has the lowest reading latency.

Research question two is the performance scalability of a voice engine with respect to increasing the number of cores in a multiprocessor. This question is addressed in Chapter three. Section 3.1 evaluates different voice-driven web architectures; the tradeoffs based on the software requirements showed that the voice engine should be installed on the server instead of being on the mobile device. Section 3.2 presents the performance evaluation results about how the voice engine scales with respect to number of cores in a multiprocessor with and without hyperthreading.

Research question three is about how to schedule virtual machines in order that the real-time application that runs inside a VM can meet its deadlines. This question is addressed in Chapter four. In this chapter, Section 4.1 presents scheduling techniques for real-time applications that run inside virtual machines which are time sharing the virtual processors. Each virtual multiprocessor has a period and an execution time that allow real-time applications to meet their deadlines that can be defined using these techniques. In Section 4.2, simulation shows the impact of the length of different virtual machine periods with and without overhead. The tradeoffs between resources consumption and period length are also discussed.

Research question four is to find a utilization based test for scheduling real-time applications on virtual multiprocessors. This question is addressed in Chapter five. This chapter presents a utilization-based test that determines if a task set is schedulable or not. If the task set is schedulable the algorithm provides the priority for each task. This algorithm avoids Dhall's effect, which may cause task sets with even very low utilization to miss deadlines.

References

[1] B. B. Brey, The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-bit Extensions: Architecture, Programming, and Interfacing. Pearson Education India, 2009.

[2] Moore's Law, "How overall processing power for computers will double every two years", http://www.mooreslaw.org.

[3] "Moore's law," Wikipedia, the free encyclopedia. 21-Feb-2016.

[4] L. Peng, J.-K. Peir, T. K. Prakash, Y.-K. Chen, and D. Koppe, “Memory performance and scalability of Intel’s and AMD’s dual-core processors: a case study,” in Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE International, 2007, pp. 55–64.

[5] J. Yan and W. Zhang, “Hybrid multi-core architecture for boosting single-threaded performance,” ACM SIGARCH Comput. Archit. News, vol. 35, no. 1, pp. 141–148, 2007.

[6] M. Bakery and R. Buyyaz, “Cluster computing at a glance,” High Perform. Clust. Comput. Archit. Syst., vol. 1, pp. 3–47, 1999.

[7] Oracle, “Brief History of Virtualization.” [Online]. Available: https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html. [Accessed: 06-Mar-2016].

[8] K. Adams and O. Agesen, “A comparison of software and hardware techniques for x86 virtualization,” ACM Sigplan Not., vol. 41, no. 11, pp. 2–13, 2006.

[9] P. Barham et al., “Xen and the art of virtualization,” ACM SIGOPS Oper. Syst. Rev., vol. 37, no. 5, pp. 164–177, 2003.

[10] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux virtual machine monitor,” in Proceedings of the Linux symposium, 2007, vol. 1, pp. 225–230.

[11] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, “Analysis of virtualization technologies for high performance computing environments,” in Cloud Computing (CLOUD), 2011 IEEE International Conference on, 2011, pp. 9–16.

[12] A. Silberschatz, H. F. Korth, S. Sudarshan, and others, Database system concepts, vol. 4. McGraw-Hill New York, 1997.

[13] R. Ramakrishnan and J. Gehrke, Database management systems. Osborne/McGraw-Hill, 2000.

[14] R. Ramakrishnan, J. Gehrke, and J. Gehrke, Database management systems, vol. 3. McGraw-Hill New York, 2003.

[15] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts. McGraw-Hill, 2011.

[16] S. S. ur Rahman, E. Schallehn, and G. Saake, “ECOS: evolutionary column-oriented storage,” in Advances in Databases, Springer, 2011, pp. 18–32.

[17] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, “Column-oriented database systems,” Proc. VLDB Endow., vol. 2, no. 2, pp. 1664–1665, 2009.

[18] D. J. Abadi, S. R. Madden, and N. Hachem, “Column-stores vs. row-stores: how different are they really?,” in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 967–980.

[19] I. Robinson, J. Webber, and E. Eifrem, Graph databases. O’Reilly Media, Inc., 2013.

[20] “Flat file database,” Wikipedia, the free encyclopedia. 16-Nov-2014.

[21] “SAP HANA,” Wikipedia. 14-Feb-2017.

[22] heise online, “SAP und HPI erhalten Deutschen Innovationspreis 2012,” heise online. [Online]. Available: http://www.heise.de/newsticker/meldung/SAP-und-HPI-erhalten-Deutschen-Innovationspreis-2012-1474254.html. [Accessed: 16-Feb-2017].

[23] F. Färber et al., “The SAP HANA Database–An Architecture Overview.,” IEEE Data Eng Bull, vol. 35, no. 1, pp. 28–33, 2012.

[24] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, “SAP HANA database: data management for modern business applications,” ACM Sigmod Rec., vol. 40, no. 4, pp. 45–51, 2012.

[25] S. Venkatraman, K. Fahd, S. Kaspi, and R. Venkatraman, “SQL Versus NoSQL Movement with Big Data Analytics,” 2016.

[26] J. Han, E. Haihong, G. Le, and J. Du, “Survey on NoSQL database,” in Pervasive computing and applications (ICPCA), 2011 6th international conference on, 2011, pp. 363–366.

[27] Y.-L. Choi, W.-S. Jeon, and S.-H. Yo, “Improving Database System Performance by Applying NoSQL.,” J. Inf. Process. Syst., vol. 10, no. 3, 2014.

[28] J. Browne, “Brewer’s CAP Theorem, 2009,” URL Httpwww Julianbrowne Comarticleviewerbrewers-Cap-Theorem, 2011.

[29] “What is Apache Cassandra?,” Planet Cassandra, 18-Jun-2015. [Online]. Available:http://www.planetcassandra.org/what-is-apache-cassandra/. [Accessed: 18-Aug-2016].

[30] “Stratio/cassandra-lucene-index,” GitHub. [Online]. Available: https://github.com/Stratio/cassandra-lucene-index. [Accessed: 18-Aug-2016].

[31] “Apache CouchDB.” [Online]. Available: http://couchdb.apache.org/. [Accessed: 18-Aug-2016].

[32] “couchbase/geocouch,” GitHub. [Online]. Available: https://github.com/couchbase/geocouch. [Accessed: 18-Aug-2016].

[33] “MongoDB for GIANT Ideas | MongoDB.” [Online]. Available: https://www.mongodb.com/. [Accessed: 18-Aug-2016].

[34] tutorialspoint.com, “MongoDB Overview,” www.tutorialspoint.com. [Online]. Available: http://www.tutorialspoint.com/mongodb/mongodb_overview.htm. [Accessed: 18-Aug-2016].

[35] “Sharding — MongoDB Manual 3.2,” https://github.com/mongodb/docs/blob/master/source/sharding.txt. [Online]. Available: https://docs.mongodb.com/manual/sharding/. [Accessed: 18-Aug-2016].

[36] J. Worsley and J. D. Drake, Practical PostgreSQL. O’Reilly Media, Inc., 2002.

[37] “Multi-node setup on Ubuntu or Debian — Citus 5.1.0 documentation.” [Online]. Available: http://docs.citusdata.com/en/v5.1/installation/production_deb.html. [Accessed: 18-Aug-2016].

[38] "PostGIS — Spatial and Geographic Objects for PostgreSQL." [Online]. Available: http://postgis.net/. [Accessed: 18-Aug-2016].

[39] "RethinkDB: the open-source database for the realtime web." [Online]. Available: https://www.rethinkdb.com/. [Accessed: 18-Aug-2016].

[40] D. Amyot and R. Simoes, "Combining VoiceXML with CCXML," in IEEE Consumer Communications & Networking Conference (CCNC 2007), 2007, pp. 342–346.

[41] B. Sprunt, L. Sha, and J. Lehoczky, "Scheduling sporadic and aperiodic events in a hard real-time system," DTIC Document, 1989.

[42] L. Abeni and T. Cucinotta, "Efficient virtualisation of real-time activities," in Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on, 2011, pp. 1–4.

[43] H. Salimi, M. Najafzadeh, and M. Sharifi, "Advantages, Challenges and Optimizations of Virtual Machine Scheduling in Cloud Computing Environments," Int. J. Comput. Theory Eng., vol. 4, no. 2, p. 189, 2012.

[44] 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.

[45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in software engineering. Springer Science & Business Media, 2012.

[46] "Research Approach." [Online]. Available: http://www.idi.ntnu.no/grupper/su/publ/html/totland/ch013.htm. [Accessed: 08-Mar-2016].

[47] N. K. Denzin and Y. S. Lincoln, Handbook of qualitative research. Sage Publications, Inc, 1994.

[48] K. Dooley, "Simulation research methods," Companion Organ., pp. 829–848, 2002.

[49] K. Y. Besedin and P. S. Kostenetskiy, "Simulating of query processing on multiprocessor database systems with modern coprocessors," in

Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on, 2014, pp. 1614–1616.

[50] R. Moussalli, I. Absalyamov, M. R. Vieira, W. Najjar, and V. J. Tsotras, "High performance FPGA and GPU complex pattern matching over spatio-temporal streams," GeoInformatica, vol. 19, no. 2, pp. 405–434, 2015.

[51] P. Huang and B. Yuan, "Mining Massive-Scale Spatiotemporal Trajectories in Parallel: A Survey," in Trends and Applications in Knowledge Discovery and Data Mining, Springer, 2015, pp. 41–52.

[52] M. Bagein, O. Pietquin, C. Ris, and G. Wilfart, "An Architecture for Voice-Enabled Interfaces over Local Wireless Networks," in Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), Orlando,(USA, FL), 2003.

[53] W. Walker, P. Lamere, and P. Kwok, "FreeTTS: a performance case study," Sun Microsyst. Inc, 2002.

[54] W. Walker et al., "Sphinx-4: A flexible open source framework for speech recognition," Sun Microsyst. Inc, 2004.

[55] L. Lundberg and S. Shirinbab, "Real-time scheduling in cloud-based virtualized software systems," in Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies, 2013, pp. 54–58.

[56] Y. Ren et al., "Residency-Aware Virtual Machine Communication Optimization: Design Choices and Techniques," in Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on, 2013, pp. 823–830.

[57] M. Kurtadikar, A. Patil, P. Toshniwal, and J. Abraham, "An inter-VM communication model supporting live migration," in Cloud & Ubiquitous Computing & Emerging Technologies (CUBE), 2013 International Conference on, 2013, pp. 63–68.

[58] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," Oper. Res., vol. 26, no. 1, pp. 127–140, 1978.

[59] I. Shin, A. Easwaran, and I. Lee, “Hierarchical scheduling framework for virtual clustering of multiprocessors,” in Real-Time Systems, 2008. ECRTS’08. Euromicro Conference on, 2008, pp. 181–190.

[60] H. Zhu, S. Goddard, and M. B. Dwyer, “Response Time Analysis of Hierarchical Scheduling: The Synchronized Deferrable Servers Approach,” in Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd, 2011, pp. 239–248.

[61] J. Y.-T. Leung and J. Whitehead, “On the complexity of fixed-priority scheduling of periodic, real-time tasks,” Perform. Eval., vol. 2, no. 4, pp. 237–250, 1982.

Chapter Two

Published as C. Niyizamwiyitira and L. Lundberg, “Performance Evaluation of SQL and NoSQL Database Management Systems in a Cluster,”International Journal of Database Management Systems (IJDMS) Vol.9, No.6, December 2017

Performance Evaluation of SQL and No SQL Database Management Systems in a Cluster

Abstract

In this study, we evaluate the performance of SQL and No SQL database management systems namely; Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB. We use a cluster of four nodes to run the database systems, with external load generators. The evaluation is conducted using data from Telenor Sverige, a telecommunication company that operates in Sweden. The experiments are conducted using three datasets of different sizes. The write throughput and latency as well as the read throughput and latency are evaluated for four queries; namely distance query, k-nearest neighbour query, range query, and region query. For write operations Cassandra has the highest throughput when multiple nodes are used, whereas PostgreSQL has the lowest latency and the highest throughput for a single node. For read operations MongoDB has the lowest latency for all queries. However, Cassandra has the highest throughput for reads. The throughput decreases as the dataset size increases for both write and read, for both sequential as well as random order access. However, this decrease is more significant for random read and write. In this study, we present the experience we had with these different database management systems including setup and configuration complexity.

Keywords

Trajectory queries, cluster computing, SQL database, NoSQL database, Cassandra, CouchDB, MongoDB, PostgreSQL, RethinkDB

2.1 Introduction

Immense volumes of data are generated continuously at a very high speed in different domains. Being unstructured and semi structured make these data heterogeneous and complex. However, efficient processing and analysis remain high priorities. The challenges include what technology in terms of software and hardware to use in order to handle these data efficiently. Processing and analysis is needed in different domains such as transportation optimization and different business analytics for telecommunication companies that seek common patterns from their mobile users in order to support business decisions.

The variety of SQL and No SQL database management systems makes it difficult to pick the most appropriate system for a specific use case. In this paper, five database systems are evaluated with respect to write and read throughput and latency. Throughput is interesting since telecom data is generated at high pace, and latency is also interesting since the speed of telecom data processing is critical.

Since big data processing requires high performance computing, we use a cluster computing environment in order to take advantage of parallel computing. We consider a case of trajectory data of mobile users.

Trajectory data represents information that describes the location of the user in time and space. A typical application of such data is that a telecommunication company wants to optimize the use of cell antennas and identify different points of interests in order to expand its business. In order to successfully process trajectory data, a proper choice of database system that efficiently respond to different queries is required.

We use trajectory data that are collected from Telenor Sverige (a telecommunication company that operates in Sweden). Mobile users' positions are tracked every five minutes for an entire week (Monday to Sunday) in a medium sized city. We are interested to know how mobile users move around the city during different hours and days of the week. This will give insights about typical behaviour in certain area at certain time. We expect periodic movement in some areas, e.g., at the location of stores and restaurants during lunch time.

Our data are spatio-temporal where at a given time t a mobile user is located at a position (x, y). The location of a mobile user is a triple (x, y, t) such that user's position is represented as a spatial-temporal point p_i with $p_i = (x_i, y_i, t_i)$.

By optimizing points of interests, different types of queries are proposed. They differ in terms of input and output:

Distance query which finds points of interests that are located in equal or less than a distance (or a radius), e.g., one kilometer from a certain position of a mobile user.

K-Nearest neighbour query that finds k nearest points of interests from a certain position of a mobile user.

Range query that finds points of interests within a space range (the range can be a triangle, polygon, …).

Region query that finds the region that a mobile user frequently passes through at certain time throughout the week.

The performance is evaluated on five open source database management systems that are capable to handle big data; Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB. We consider random access requests as well as sequential requests. The hardware is a cluster with four nodes that run the database, with four external load generators for random workloads, and one load generator for sequential workloads. By using this kind of data, an operator knows the locations that are the most, or the least visited during a certain time. Therefore, in order to avoid overloading and underloading at such locations, antenna planning can be updated accordingly. For business expansion, a busy location during lunch time is for instance good for putting up a restaurant.

The rest of the paper is organized as follows; Section 2 defines trajectory data concept, Section 3 summarizes related work, Section 4 gives an overview of database management systems, Section 5 describes the methodology, Section 6 presents the results. Section 7 presents some discussions and analysis, and finally Section 8 draws conclusions.

2.2 Trajectory Data [1]

2.2.1 Definition of trajectory

A trajectory is a function from a temporal domain to a range of spatial values, i.e., it has a start and end time during which a space has been travelled (see Equation 1) [2].

$$[t_{begin}\ t_{end}] \rightarrow space \tag{1}$$

A complete trajectory is characterized by a list of triples $p = (x, y, t)$, thus a trajectory is defined as a sequence of positions T_{pos}.

$$\mathrm{T}_{pos} = \{p_1, p_2, \dots, p_n\} \tag{2}$$

where $p_i = (x_i, y_i, t_i)$ represents a spatio-temporal point, Figure 1 shows a trajectory.

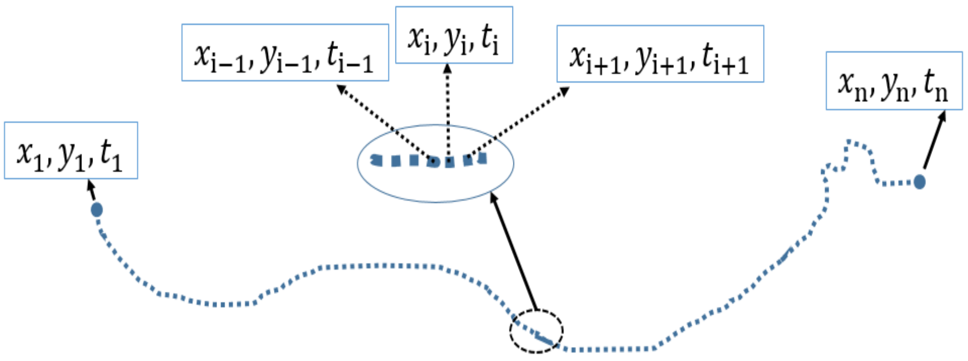


Figure 1. Mobile user's trajectory as a sequence of triples

In this study, the trajectory data space is represented by latitude and longitude; x represents latitude and y represents longitude, and time is represented by t.

2.2.2 Data Description

A location update is generated when a handset is generating traffic either by downloading or uploading data. The data used in this paper are collected every five minutes for an entire week in a medium sized city, i.e., the data is at rest.

This data is anonymized in order to comply with the company agreement about undisclosuring users' information.

Table 1. Mobile user data description

Mobile User's attributes		**Short description**
1	User ID	User identification
2	Site ID	Identification number of the site location
3	Weekday	Day of the week that the data has been recorded
4	Time	Clock Time that the data has been recorded
5	Profile ID	The user profile identification such as a salesperson, a store, with a mobile that runs a certain operating system like android or any other
6	Segment ID	Type of client such as Corporate client, Cost Aware, Quality Aware
7	SourceGSM	Used network technology is Global System for Mobile communication (GSM)
8	SourceUMTS	Used network technology is Universal Mobile Telecommunications System (UMTS)
9	SourceLTE	Used network technology is Long Term Evolution (LTE)
10	Easting	User's position with respect to east
11	Northing	User's position with respect to north
12	Latitude decimal	User's location in terms in latitude coordinates

13	Longitude decimal	User's location in terms in longitude coordinates
14	Cell municipality	Municipality of the Cell antenna's location
15	Cell county	County of the Cell antenna's location
16	Cell city	Cell antenna's location in terms of City
17	Cell postcode	Postcode of the Cell antenna's location
18	Cell address	Address of the Cell antenna's location

We have three datasets with different sizes;

1. Dataset0 has 6,483,398 records and 18 attributes,
2. Dataset1 has 12,966,795 records and 18 attributes,
3. Dataset2 has 25,933,590 records and 18 attributes.

Dataset2 has the biggest size, it is four times Dataset0 or two times Dataset1, the dataset size was scaled until the available resources for the experiment, the cluster memory was maximized by the size, thus we stopped at dataset2. Table 1 shows the 18 attributes in each data record used by Telenor.

2.2.3 Definition of Trajectory Queries

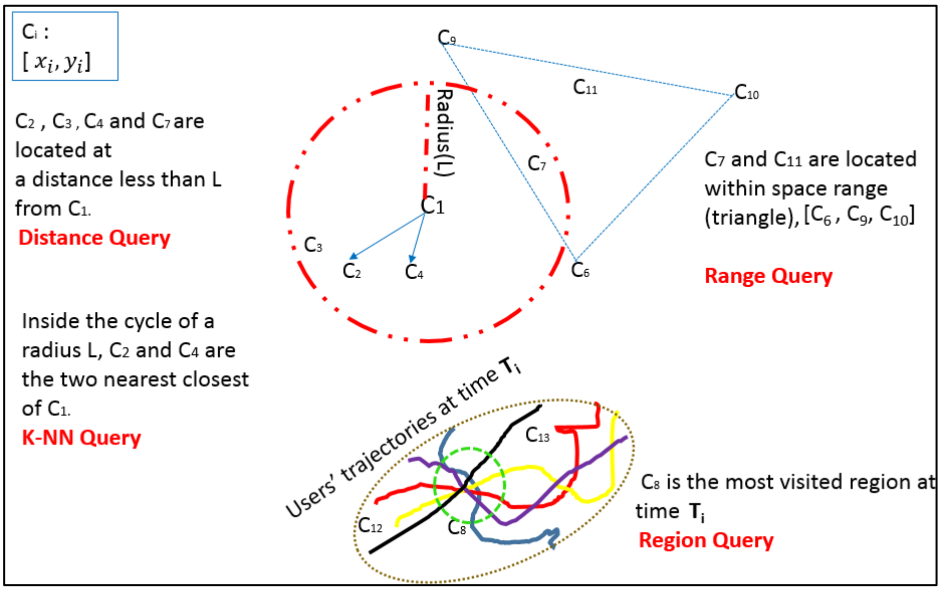
Trajectories queries on spatio-temporal data are the foundation of many applications such as traffic analysis, mobile user's behaviour, and many others [3] [4]. In the context of location optimization, common trajectory queries that we consider in this study are: Distance query, *k*-nearest neighbour query, Range query, and Region query. We will describe these queries in the subsections below.

Figure 2 visualizes the four query types; C_i is the location of cell i, each C_i is represented by (x_i, y_i) where x_i is latitude and y_i is longitude. A distance query returns a list of cells that are located at a certain distance from a location, e.g., within distance L from the position of C_1. The query returns the list [C_2, C_3 , C_4, C_7].

We can find the two cells that are closest to cell C_1, by using a k-nearest neighbour query with $k = 2$.

Given a triangular space, a range query returns the cells that belong to that space.

A region query returns the cell that is the most frequently visited at a certain time. e.g., cell C_8 at time t_i (see Figure 2).



C_i : $[x_i, y_i]$
C_2 , C_3 , C_4 and C_7 are located at a distance less than L from C_1.
Distance Query
Inside the cycle of a radius L, C_2 and C_4 are the two nearest closest of C_1.
K-NN Query
Radius(L)
C_1
C_2
C_3
C_4
C_6
C_7
C_9
C_{10}
C_{11}
C_7 and C_{11} are located within space range (triangle), $[C_6, C_9, C_{10}]$
Range Query
Users' trajectories at time T_i
C_{12}
C_8
C_{13}
C_8 is the most visited region at time T_i
Region Query

Figure 2. Visualization for Query Types

2.2.4 Distance Query

Definition: A distance query returns all the cells in the circle whose distance from a given position is less than a threshold [3], [4]. Figure 3 shows inputs and output of a distance query.

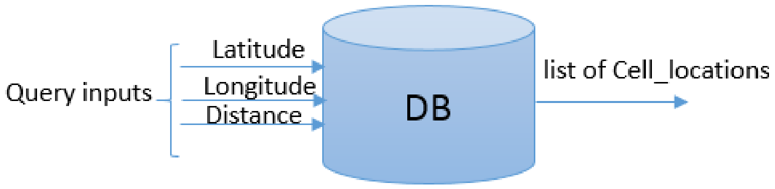


Figure 3. Distance Query

2.2.5 *k*-Nearest Neighbour Query

Definition: A *k*-Nearest Neighbour (*k*-NN) Query returns from zero up to *k* cells which are the closest to a given position (x, y) [5] [6]; the *k* results are ordered by proximity. *k*-NN is bounded by a distance, and if *k* cells within the given distance from the given position is not indicated, the query behaves like a distance query.

Figure 4 shows the inputs and output of a *k*-NN query.

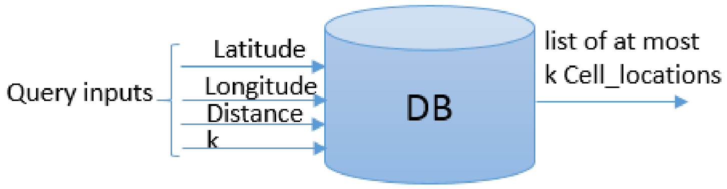


Figure 4. *k*-NN query

2.2.6 Range Query

Definition: Range query returns all the cells that are located within a certain space shape (polygon) [3]. In this paper we only consider triangles. Figure 5 shows inputs and output of range query to find cells that are in the triangle. In this paper, the range query retrieve cells that are located inside a triangle. The triangle space is defined by nodes latitude and longitude points.

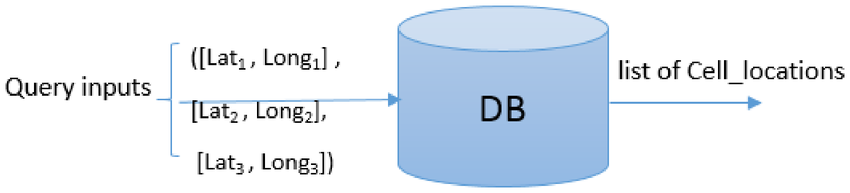


Figure 5 Range query

2.2.7 Region Query

Generally, trajectories of mobile users are independent of each other. However, they contain common behaviour traits such as passing through a region at a certain periods, e.g., passing through the shopping center during lunch time.
Definition: A region query identifies the cell that is the most visited at a given point in time [3].

Figure 6 shows inputs and output of a region query to find the cell that is the most visited at time T_i. In this paper, the region query takes time as input, then at a certain fixed time users are moving around the city, region query picks up the region that most users are located.

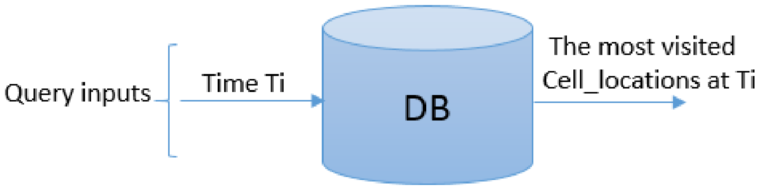


Figure 6. Region query

2.3 Related Work

In [7], the authors proposed an approach and implementation of spatio-temporal database management systems. This approach treats time-changing geometries, whether they change in discrete or continuous steps. The same approach can be used to tackle spatio-temporal data in other database management systems. We evaluate trajectory queries on existing general purpose database management systems (Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB). In [8], the author describes requirements for database management systems that support location-based service for spatio-temporal data. A list of ten representative queries for stationary and moving reference objects is proposed. Some of those queries are related to the queries considered in Section 2.

In [9], Dieter studied trajectory moving point objects, he explained three scenarios, namely constrained movement, unconstraint movement and movement in networks. Different techniques to index and to query in these scenarios define their respective processing performance. The author modelled a trajectory as triple (x, y, t), we use the same model in this study.

In [10], the authors introduced querying moving objects (trajectory) in SECONDO, a DBMS prototyping environment particularly geared for extension by algebra modules. The querying is done using an SQL-like language. In our study, we are querying moving object using SQL and Not Only SQL (NoSQL) querying languages on top of different database management systems. The authors provide a benchmark on range queries and nearest neighbour queries in SECONDO DBMS for moving data object in Berlin. The moving object data were generated using computer simulation based on the map of Berlin [11]. This benchmark could be extended to other queries such as region queries, distance queries, and so on. In our study, we apply these queries on real world trajectory data, i.e., mobile users' trajectory from Telenor Sverige.

In [5], the authors introduced a new type of query, Reverse Nearest Neighbour (RNN) which is the opposite to Nearest Neighbour (NN). RNN can be useful in applications where moving objects agree to provide some kind of service to each other, whenever a service is needed it is requested from the nearest neighbour. An object knows objects that it will serve in the future using RNN. RNN and NN are represented by distance query in our study.

In [12], the authors studied an aggregate query language for GIS and no-spatial data stored in a data warehouse. In [13], the authors studied k-nearest neighbour search algorithm for historical moving object trajectories, this k-nearest neighbour is one of the queries that is considered in our study.

In [14], the authors presented techniques for indexing and querying moving object trajectories. These data are represented in three dimensions, where two dimensions correspond to space and one dimension corresponds to time. We also represent our data in 3D as (x, y, t), with x, y represent space whereas t represents time.

Query processing on multiprocessors was studied in [15], the authors implemented an emulator; this is a software that uses computing cluster with NVIDIA GPUs or Intel Xeon Phi coprocessors for relational query processing of a parallel DBMS on a cluster of multiprocessors. This study is different from ours in a sense that we evaluate query processing on real physical hardware with existing general purpose database management systems. Query processing using FPGA and GPU on spatial-temporal data was studied in [16]. The authors presented a FPGA and GPU implementation that process complex queries in parallel, the study did not investigate the performance on various existing database systems, a distributed environment was also not considered. In our study we investigate query processing on various database management systems on a cluster. In [17], the authors conducted a survey on mining massive-scale spatio-temporal trajectory data based on parallel computing platforms such as GPU, MapReduce and FPGA, again existing general purpose database systems were not evaluated. The authors presented a hardware implementation for converting geohash codes to and from longitude/latitude pairs for spatio-temporal data [18], the study shows that longitude and latitude coordinates are the key points for modelling spatio-temporal data. In our paper, we also use these coordinates for location based querying. The benchmark for NoSQL databases, namely Apache Cassandra, Couchbase, HBase, and MongoDB is presented in [19]. This benchmark was performed on Amazon Web Service (AWS) EC2 instances. They used Yahoo! Cloud Serving Benchmark (YCSB) data. In terms of throughput and horizontal scalability, Cassandra is the best, Hbase is the second, CouchBase is the third and MongoDB is the fourth. In this paper we have not considered

CouchBase and HBase, since they are in memory databases, i.e., they used direct memory which is good for processing small data in real-time. Therefore this would be smaller for our workload. We used Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB on real-world workload instead simulated workload.

2.4 Database Management System Overview

The presence of unstructured data stimulated the invention of new databases, since Relational Database Management Systems (RDBMS) that uses Structured Query Language (SQL) cannot handle unstructured data efficiently. A new data model, Not Only SQL (NoSQL) was introduced to deal also with unstructured data [20]. The main features of NoSQL follow the CAP theorem (Consistency, Availability, and Partition tolerance). The core idea of CAP is that a distributed system cannot meet these three needs simultaneously (see Figure 7). Depending on the data models, NoSQL can be relational, key value based, column based, and document based. In this study we choose five open source databases that have diverse features of SQL (PostgreSQL) and NoSQL (Cassandra, CouchDB, MongoDB and RethinkDB).

A key value data model means that a value corresponds to a key, in column based systems data are stored by column, each column is an index of the database, queries are applied to columns, whereby each column is treated one by one. A document-based database stores in the JSON or XML format, each document, is indexed and it has a key.

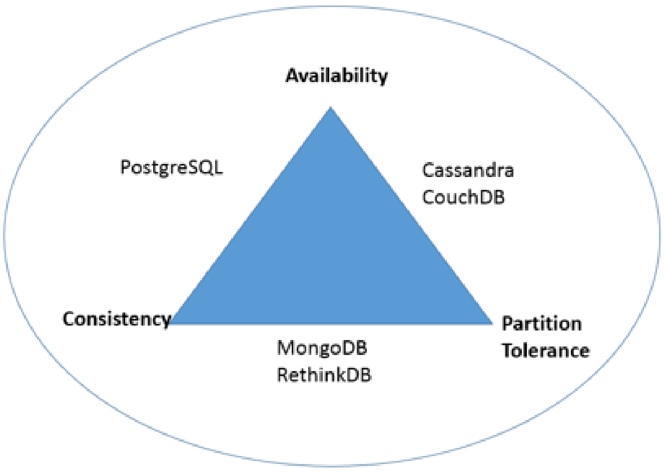


Figure 7. Principles of Distributed Database systems.

2.4.1 Cassandra

Apache Cassandra is an open-source NoSQL column based database. It is written in Java, it is a top level Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable. It is a distributed database for managing large amounts of structured data across many commodity servers, while providing highly available service with no single point of failure. In CAP, Cassandra has availability and partition tolerance (AP) with eventual (delayed) consistency. Cassandra offers continuous availability, linearly scaling performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones. Cassandra has a masterless ring architecture [21]. The keyspace is similar to database in RDBMS, inside keyspace there are tables which are similar to tables in RDBMS, column and rows are similar to those of RDBMS' tables. The querying language is Cassandra Query Language (CQL) that is very similar to SQL [22]. Cassandra does not natively support spatial indexing but this can be extended via Stratio's Cassandra Lucene index. Stratio's Cassandra Lucene Index is an additional module for Apache Cassandra, it extends its index functionality to provide near real time search for text search, field based sorting, and spatial index. [23].

2.4.2 CouchDB

CouchDB is written in Erlang and it stores data as JSON documents. Access documents and query indexes with a web browser, via HTTP. CouchDB indexes, combines, and transforms documents with JavaScript. It is highly available and partition tolerant, but also eventually consistent, CouchDB supports masterless setup [23]. The system does not natively support spatial queries, we add a module GeoCouch for spatial index on CouchDB [25].

2.4.3 MongoDB

MongoDB is an open-source NoSQL document database, it is written in C++. MongoDB has a database, inside the database there are collections, like tables in RDBMS. Inside a collection there are documents, these are like a tuple/row in RDBMS, and inside a document there are fields which are like columns in RDBMS [25][26]. MongoDB is consistent and partition tolerant. MongoDB has natively a built in function for spatial queries and it has a sharding (separating very large database into smaller, faster and easily manageable parts called shards across cluster's nodes) feature to support horizontal scalability of the database in master/slave fashion [27].

2.4.4 PostgreSQL

PostgreSQL is an open source object RDBMS written in C that has two features according to the CAP theorem; those are availability, i.e., each user can always read and write, and consistency, i.e., all users have the same view of data. PostgreSQL organises data in columns and rows [28, p. 3]. PostgreSQL does not natively support horizontal scalability as well as spatial queries, PostgreSQL is extended by CITUS and PostGIS to support scalability in master/slave fashion and spatial queries indexing respectively [29][30].

2.4.5 RethinkDB

RethinkDB is an open source NoSQL database system. RethinkDB is written in C ++, it is horizontally scalable in a master/slave setup, it is mostly designed to facilitate real-time updates for query results to applications [31].

RethinkDB natively support spatial queries using GeoJson. The system uses the ReQL query language that is available for Python, Ruby, and Java.

2.5 Methodology

This section presents details about the experimental setup, hardware and software; the measurement procedure is also explained. All the data are represented in the comma separated values (CSV) format.

2.5.1 Experiment setup

A cluster is made up of 4 nodes, each node is Dell powerEdge R320 with operating system: Ubuntu 14.04.3 LTS x86_64. Each node has 23 GB RAM, disk size of 279.4 GB, and a processor (Intel(R) Xeon(R) CPU E5-2420 v2) with 12 cores, each core is hyperthreaded into 2 cores, which gives 24 virtual cores. These servers run Java development kit jdk 1.8.0.72. These servers are only running our database management systems, nothing else. Another machine (called load generator) with the same features outside of this database cluster generates the load for sequential writing and reading towards the cluster. This setup is described in Figure 8. We use four load generators for random writing and reading, each of these generators has also the same features mentioned earlier. Figure 9 shows the setup for random load.

Cassandra 3.0.3 is installed at each of the nodes in the cluster in a ring topology with each node has the same role as the other, i.e. master/master fashion. Stratio Lucene is installed and connected to Cassandra at each of the nodes. The replication factor equals the number of nodes, i.e., every node has the same copy of data [32]. The consistency level is Quorum, i.e., return most recent data from a majority of replicas [34].

CouchDB is also installed on each of the cluster nodes in master/master fashion, after that Geocouch is installed and connected to CouchDB [23][24].

MongoDB is installed on the cluster in master/slave fashion, where the master mongos (mongo master server) is installed on one of the nodes, and the three config servers which act like slave servers are installed on the remaining three nodes [35][34].

PostgreSQL is also installed with PostGIS on each of the cluster nodes. After that, CITUS is connected to PostgreSQL in order to support the distribution, and as a result PostgreSQL becomes a distributed database system in master/slave fashion where one of the nodes acts as the master and the others as the slaves [30][35].

RethinkDB is also installed at each of the cluster nodes in a master/slave fashion; one node is a master [31].

Table 2 shows the details of different features of the database management systems that we are evaluating. In Table 2, BASE is Basically Available, Soft state, Eventual consistency, and ACID is Atomicity, Consistency, Isolation, and Durability.

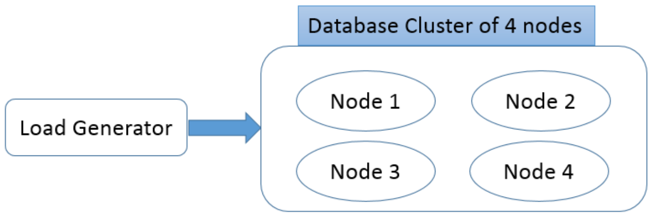


Figure 8. Experiment setup for sequential workload

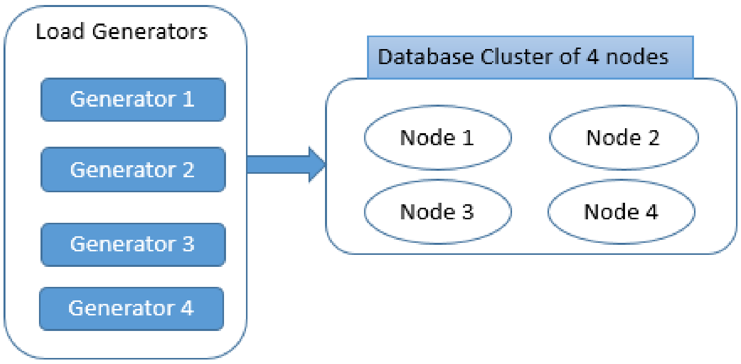


Figure 9. Experiment setup for random workload

Table 2. Database systems features

	Cassandra	**CouchDB**	**MongoDB**	**PostgreSQL**	**RethinkDB**
Extension module to support Spatial query	Stratio's Cassandra Lucene Index	GeoCouch	Natively support Spatial query	PostGIS	Natively support Spatial query
Extension module to support distributed computing	Natively distributed		Natively distributed	CITUS	Natively distributed
Language	Java	Erlang	C++	C	C++
Query Language	Cassandra Query Language (CQL)	Json	Json	Structure Query Language (SQL)	RethinkDB query Language (ReQL in python)

Partitioning method	Sharding	Sharding	Sharding	Sharding	Sharding
Replication method	Master/Master	Master/Master	Master/slave	Master/slave	Master/slave
Transaction property	BASE	BASE	BASE	ACID	BASE
Version	3.0.3	1.6.1	3.0.9	9.5.3	2.3.1
License	Apache 2.0	Apache 2.0	GNU AGPL v3.0	PostgreSQL	AGPL
Data Model	Column + key	Document + Key	Document + Key	Column + row (relational DBMS)	Document + Key
First Release	2010	2005	2007	1986	2009

2.5.2 Write procedure

Two ways of writing are considered; sequential and random order. During sequential writing, the workload is generated by one load generator machine towards the cluster and the given entire dataset is written in sequential order.

For random writing, there are four load generators, each generator contains the same dataset. Each generator makes a write request that writes a quarter of the entire dataset size records in a random order into the database cluster.
For Cassandra, data into CSV format is imported into the Cassandra database cluster. The node that receives the request will get a list of N nodes responsible for replicas of the keys in the write request from ReplicationStrategy. It then sends a RowMutation message to each of the N nodes. Then each node append the write to the commit log and update the in-memory Memtable data structure with the write. All the writes either random or sequential, are written using SSTable loader.
For CouchDB, the CSV data format is imported from the load generator into CouchDB using a couch import script that is written in Node.js. After writing, each row in the source file becomes a document. In CouchDB, every node in

the cluster participates in the importing and writing. Writing are done using http_bulk.
For MongoDB, CSV data format is imported using mongo import script towards mongos (master). Each row becomes a document, thereafter mongos shards the data across the cluster.
For PostgreSQL, CSV data format is imported and written on the master, which shards it across the cluster nodes.
For RethinkDB, CSV data format is imported and written to the database through the master node which shards it across the cluster.
When the writing is completed, the message that indicates how many records are written during how long time, will appear on the load generator machine (s).
Each record (row) in all datasets have the same size, i.e., each row is 7.94 KB in CSV format.

2.5.3 Reading procedure

Reading is also conducted in two ways; sequential and random. In the sequential procedure, the read request is generated from one load generator, each of the queries, distance, *k*-nearest neighbour, range, and region is sent 10 times to each of the database systems (Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB).
For random read, read request, i.e., queries, are generated from 4 load generators, and the responses are gathered at the load generators.

2.5.4 Measurement procedure

In this study, we measure the write and read latency, as well as the throughput. The latency measured in these experiments shows how long each individual write or read request takes to be processed. It does not include network latency between the load generator and the database cluster. Instead, it is measured from the database perspective, i.e., the time that is required to process a single request.

The write latency is the number of milliseconds required to fulfill a single write request. The time period starts when one of the cluster nodes receives write request from the load generator, and ends when the nodes complete a write request.

The read latency is the number of milliseconds required to fulfill a read request. The time period starts when one of the cluster nodes receives read request from the load generator, and ends when the node completes a read request. All the results are the average of ten runs. Measurements are conducted on three datasets of different size, namely dataset0, dataset1, and dataset2 as defined in Section 2. Read latency is measured with respect to the four different queries as defined also in Section 2.

For the write latency, the workload is 100% write only, and for read latency, the workload is 100% read.

We measure the throughput as operations per second,
Throughput = number of completed operations / time to complete those operations.
The latency is measured on the database cluster servers, whereas the throughput is measured on load generators.

2.6 Results

All datasets are populated into Cassandra and PostgreSQL, and CouchDB without any transformation. Whereas, in MongoDB and RethinkDB, coordinates attributes (latitude and longitude) were combined into an array location attribute in order to be able to use spatial function in MongoDB and RethinkDB. Results for dataset0 and dataset1 are presented as tables in the appendix. Whereas results from dataset2 are plotted in this section.

Results in figures 10 and 11 show the write throughput and latency for dataset2 for both sequence and random. Figures 12-19 show the read throughput and the latency for distance, k-NN, range, and region queries in dataset2. The throughput is measured as operations per second whereas the latency is presented in milliseconds.

During the write workload, MongoDB, PostgreSQL, and RethinkDB use one master node only therefore, their latency and throughput are consistent throughout all nodes. However, Cassandra and CouchDB scale their throughput as the number of nodes increases. CouchDB has a slightly inconsistent latency as the number of nodes increases, i.e., the latency goes up and down a little as the number of nodes increases.

The dataset size impacts the throughput negatively, especially for random write since the search grows as the dataset size increases. Therefore, the throughput decreases and the latency increases (see Appendix and figures 10(b) and 11(b)).

During read, Cassandra has in most cases the highest throughput, PostgreSQL has the second highest throughput, and MongoDB has the third whereas CouchDB has the lowest. The read throughput scales up as the number of nodes increases for all the database systems, see figures 12, 14, 16, and 18. Generally, MongoDB has the lowest read latency for higher number of nodes, see figures 13, 15, 17, and 19.

For one node, the read latency for Cassandra, MongoDB, and PostgreSQL are similar, and they are significantly lower than CouchDB and RethinkDB.

Generally, Cassandra has the highest read throughput. MongoDB and PostgreSQL have almost similar read throughput following Cassandra. RethinkDB has the fourth highest throughput, whereas CouchDB has the lowest throughput. This is shown in figures 12, 14, 16, and 18 for different queries. Database systems installed into master/slave fashion exhibit immediate writing consistency, e.g., MongoDB, PostgreSQL, and RethinkDB. Whereas those installed into master/master fashion present eventual consistency, e.g., Cassanda and CouchDB.

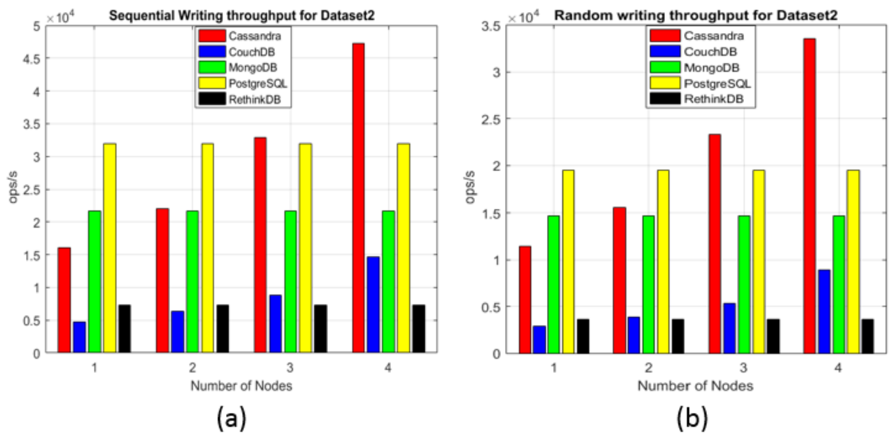


Figure 10. (a) Sequential write throughput for dataset2, (b) Random write throughput for dataset2

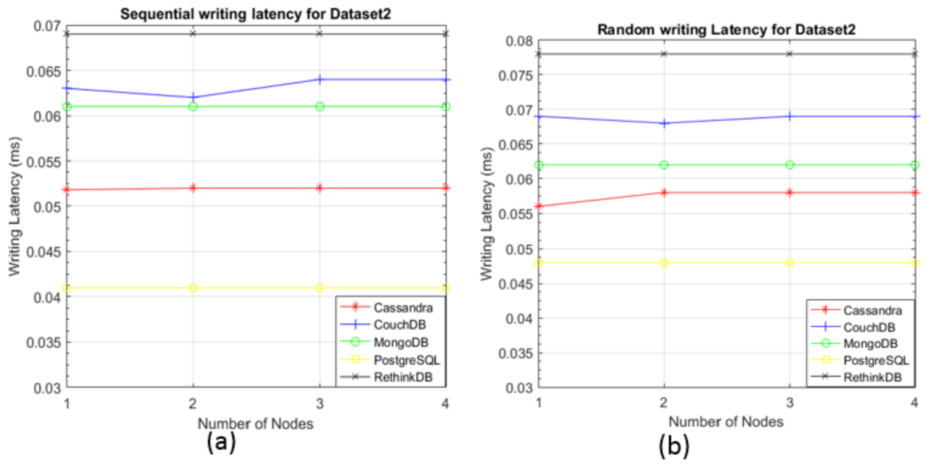


Figure 11. (a) Sequential write latency for dataset2, (b) Random write latency for dataset2

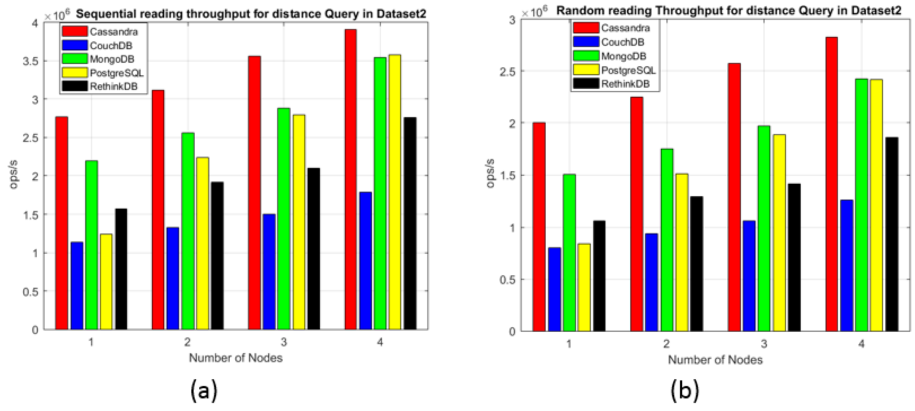


Figure 12. (a) Sequential read throughput for distance query in dataset2, (b) Random read throughput for distance query in dataset2

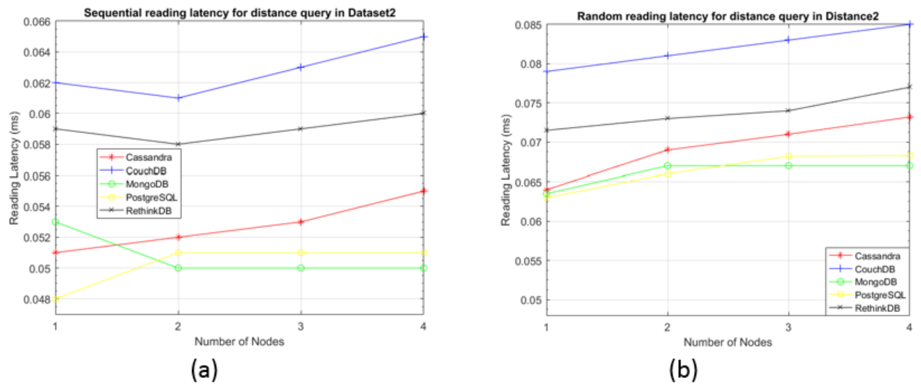
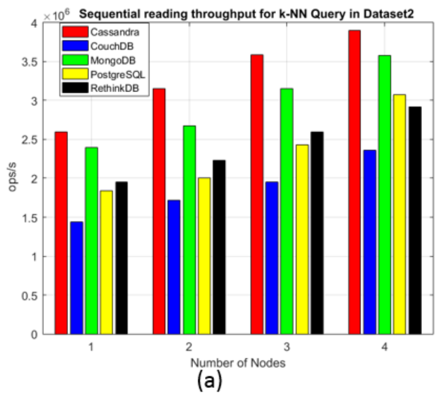


Figure 13. (a) Sequential read latency for distance query in dataset2, (b) Random read latency for distance query in dataset2



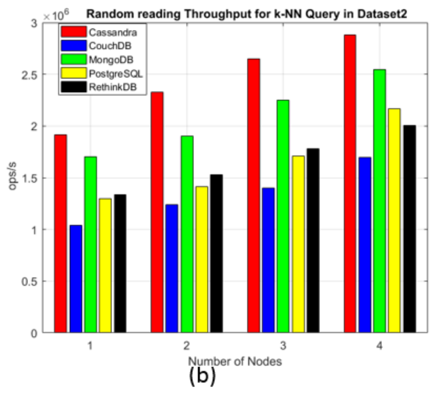
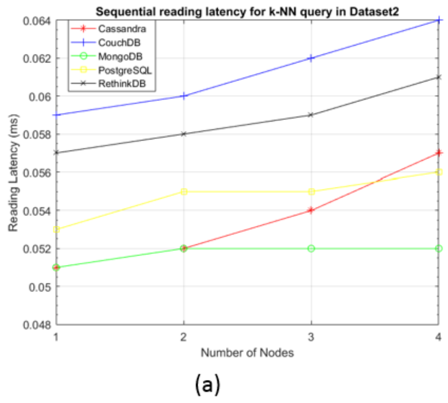


Figure 5. (a) Sequential read throughput for *k*-NN query in dataset2, (b) Random read throughput for *k*-NN query in dataset2



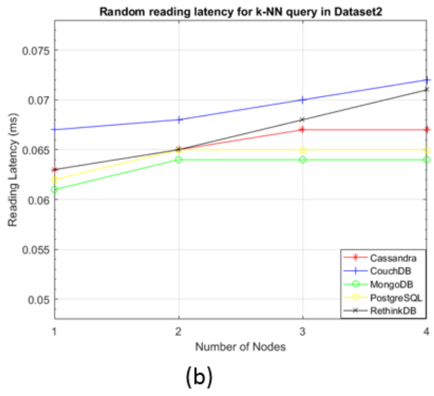
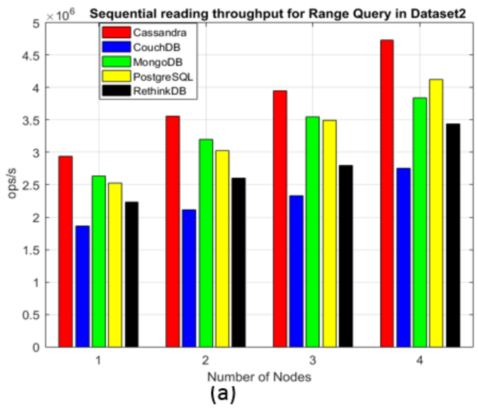


Figure 6. (a) Sequential read latency for *k*-NN query in dataset2, (b) Random read latency for *k*-NN query in dataset2



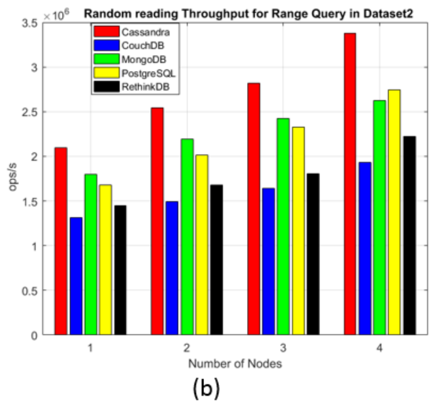
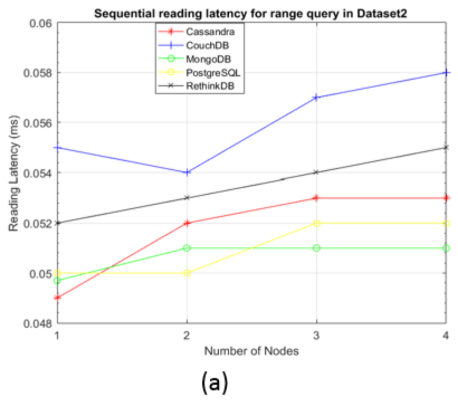


Figure 16. (a) Sequential read throughput for Range query in dataset2, (b) Random read throughput for Range query in dataset2



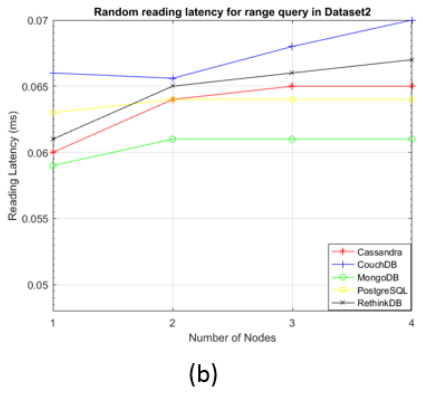


Figure 17. (a) Sequential read latency for range query in dataset2, (b) Random read latency for range query in dataset2

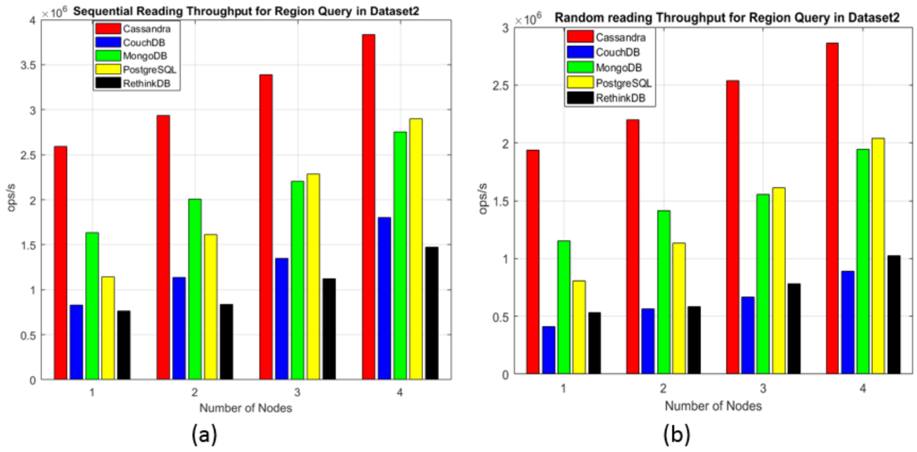


Figure 7. (a) Sequential read throughput for region query in dataset2, (a) Random read throughput for region query in dataset2

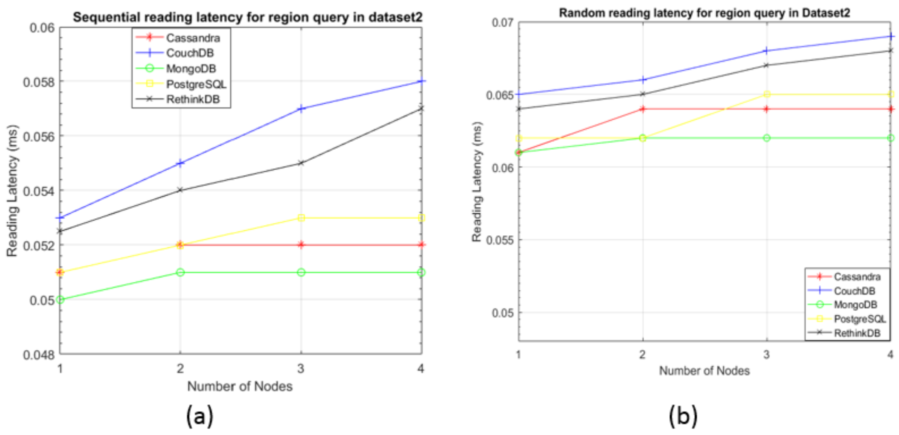


Figure 19. (a) Sequential read latency for region query in dataset2, (b) Random read latency for region query in dataset2

2.7 Discussion and Analysis

In terms of scalability, Cassandra outperforms the other database systems throughout our experiments. Cassandra shows lower write latency for one node and it slightly increases for two nodes, then it stays stable for more nodes. Cassandra does not presents the best write and read latency, but it has the highest throughput, this shows that Cassandra has more parallelism.

PostgreSQL presents the lowest write latency, followed by MongoDB, which is followed by Cassandra, CouchDB, and RethinkDB which has the highest write latency. In general, Cassandra has the highest write throughput as the number of nodes increases whereas RethinkDB has the lowest throughput.

CouchDB is scalable, however, the slowest in reading and its reading throughput is similarly affected. We observe that Cassandra and CouchDB present similar speed up, However, Cassandra has higher throughput than CouchDB.The reason is that CouchDB serves mainly as backend to serve the web whereby retrieving a lot of records at the same time may become very slow. Usually, there will be cashing functions and closest region hosting that will support CouchDB when it is backing the web. The writing and reading throughput of CouchDB is not as high as expected; this is caused by fetching data over HTTP protocol which is essentially a high latency protocol. Therefore, CouchDB scalability does not exploit the parallelism by achieving higher throughput as expected. MongoDB, PostgreSQL, and RethinkDB are installed in master slave fashion, thus they are not scalable for write since only the master node writes.

PostgreSQL presents the best write latency and the highest throughput. From one node until three nodes PostgreSQL outperforms other database systems. This is due to delayed commit that happens at the end of whole dataset write workload, thus speeding up the writing. Introducing an explicit commit after each record (row) insert could slow down PostgreSQL significantly. Delayed commit is usually the default in PostgreSQL, it may cause data loss in case of database crash, and therefore it should not be used for very sensitive data like bank transactions.

MongoDB has in most cases the lowest read latency because it has a spatial function that quickly process spatial queries. MongoDB is fast for reads because it shards data across nodes, when a query is launched, only the concerned nodes will respond to the query. This avoid going over the whole dataset. The same principle is also applied in PostgreSQL with the help of CITUS, the extension that horizontally scales PostgreSQL across commodity servers using sharding.

Since RethinkDB is designed for real-time applications such as game live score and online multiplayer games, during which writing must be acknowledged by the server and subsequently are available to the client. Such data are in most cases relatively small in order to be processed at low latency, due to big size datasets that are used in our experiments, RethinkDB suffers from high latency and low throughput. However, the performance is better for read than write.

As expected, sequential processing has higher throughput and a little shorter latency than random processing. Increase of the dataset size causes the throughput to decrease significantly, this is a result of the overhead that becomes higher as the dataset's size increases. The latency is also affected by the increase of the dataset in such a way that the latency becomes a bit higher. However, this increase is not significant. This is intuitively true since the throughput is measured from the load generator, and latency is measured from the database servers.

The decrease of throughput with the increase of the dataset's size is even more noticeable for random writing and reading. Random writing and reading has lower throughput comparing to sequential writing and reading for dataset0, dataset1, and dataset2 respectively. This is caused by the overhead due to the random search order across the entire dataset, thus the search becomes exhaustive for both writing and reading. We saw a throughput decrease of almost 10% from sequential to random processing.

In order to get better results, we have used new hardware and we have used workload of different sizes. We considered also random and sequential processing for both write and read. Compared to the benchmarking of Cassandra, Couchbase, HBase, and MongoDB [19], we have similar trends where Cassandra outperforms MongoDB. Moreover, we include the comparison for five database systems, Cassandra, CouchDB, MongoDB, PostgreSQL and RethinkDB. Our results can serve as benchmark for other studies.

2.8 Conclusions

In this paper, we evaluate the write and read throughput as well as the latency of five SQL and NoSQL database management systems namely; Cassandra, CouchDB, MongoDB, PostgreSQL, and RethinkDB. The evaluation is conducted on a cluster using telecommunication data collected from Telenor Sverige. We did measurements on three datasets of different sizes; dataset0, dataset1, and dataset2. We measured the write throughput and latency of each of the datasets, and the read throughput and latency for four queries, namely distance, k-nearest neighbour, range, and region queries. Both writing and reading are experimented in sequential, and in random on a database cluster system of four nodes.

For read queries, all database management systems are scalable as the number of nodes increases. However, only Cassandra and CouchDB show scalability for data writing. It is observed that as the dataset's size increases the throughput decreases and the latency increases.

The write throughput results show that on a single server, PostgreSQL performs better than others whereas Cassandra exhibits the highest throughout for higher number of nodes. PostgreSQL also presents the lowest latency for all writes. The reading results for all four queries show that Cassandra has the highest throughput even though it does have the lowest latency. This is a result of more parallelism in Cassandra. CouchDB has the lowest read throughput and highest latency though it is scalable, i.e., as the number of nodes increase the throughput increases as well.

In our experiments, we observed that in some cases PostgreSQL when featured by CITRUS, shows lower reading latency and horizontal scalability features than MongoDB, CouchDB, and RethinkDB. If the data to be processed requires the flexibility of traditional relational database (SQL), PostgreSQL would be preferred, if scalability matters, one would choose Cassandra. MongoDB, CouchDB, and RethinkDB would favour data that are transferred over the web, since they are document oriented that are easy to interpret for the web.

During our experiments, we experienced installation challenges of different database management systems. In terms of installation, Cassandra was straight forward except that spatial query extension that was challenging to be incorporated into the system. CouchDB was the most challenging, it took more time than the other database systems, especially installing it in a distributed fashion on many nodes. MongoDB was also straight forward with sharding that was a bit challenging. PostgreSQL was straight forward. However incorporating the distributing platform CITUS was challenging. RethinkDB was the easiest to install. By ranking these database systems according to easiness of installation, RethinkDB is the first, MongoDB is the second, Cassandra is the third, PostgreSQL is the fourth, and CouchDB is the fifth.

As far as mobile users' data analytics is concerned, since the processing and analysis is not done on the fly as the data come in, immediate consistency is not a big issue. Hence Cassandra would suits to process it, because it has high throughput and a relatively low latency with eventual consistency and availability across the cluster.

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. We also thank Telenor Sverige for providing the data.

References

[1] C. Niyizamwiyitira and L. Lundberg, "Performance Evaluation of Trajectory Queries on Multiprocessor and Cluster," in Computer Science & Information Technology, Vienna,Austria, 2016, vol. 6, pp. 145–163.

[2] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot, "A conceptual view on trajectories," Data Knowl. Eng., vol. 65, no. 1, pp. 126–146, 2008.

[3] Y. Zheng and X. Zhou, Computing with spatial trajectories. Springer Science & Business Media, 2011.

[4] N. Pelekis and Y. Theodoridis, Mobility data management and exploration. Springer, 2014.

[5] R. Benetis, C. S. Jensen, G. Karčiauskas, and S. Šaltenis, “Nearest neighbor and reverse nearest neighbor queries for moving objects,” in Database Engineering and Applications Symposium, 2002. Proceedings. International, 2002, pp. 44–53.

[6] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, “Nearest neighbor search on moving object trajectories,” in Advances in Spatial and Temporal Databases, Springer, 2005, pp. 328–345.

[7] M. Erwig, R. H. Gu, M. Schneider, M. Vazirgiannis, and others, “Spatio-temporal data types: An approach to modeling and querying moving objects in databases,” GeoInformatica, vol. 3, no. 3, pp. 269–296, 1999.

[8] Y. Theodoridis, “Ten benchmark database queries for location-based services,” Comput. J., vol. 46, no. 6, pp. 713–725, 2003.

[9] D. Pfoser, “Indexing the trajectories of moving objects,” IEEE Data Eng Bull, vol. 25, no. 2, pp. 3–9, 2002.

[10] V. T. De Almeida, R. H. Güting, and T. Behr, “Querying moving objects in secondo,” in null, 2006, p. 47.

[11] C. Düntgen, T. Behr, and R. H. Güting, “BerlinMOD: a benchmark for moving object databases,” VLDB J., vol. 18, no. 6, pp. 1335–1368, 2009.

[12] L. I. Gómez, B. Kuijpers, and A. A. Vaisman, “Aggregation languages for moving object and places of interest,” in Proceedings of the 2008 ACM symposium on Applied computing, 2008, pp. 857–862.

[13] Y.-J. Gao, C. Li, G.-C. Chen, L. Chen, X.-T. Jiang, and C. Chen, “Efficient k-nearest-neighbor search algorithms for historical moving object trajectories,” J. Comput. Sci. Technol., vol. 22, no. 2, pp. 232–244, 2007.

[14] D. Pfoser, C. S. Jensen, Y. Theodoridis, and others, “Novel approaches to the indexing of moving object trajectories,” in Proceedings of VLDB, 2000, pp. 395–406.

[15] K. Y. Besedin and P. S. Kostenetskiy, “Simulating of query processing on multiprocessor database systems with modern coprocessors,” in Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on, 2014, pp. 1614–1616.

[16] R. Moussalli, I. Absalyamov, M. R. Vieira, W. Najjar, and V. J. Tsotras, “High performance FPGA and GPU complex pattern matching over spatio-temporal streams,” GeoInformatica, vol. 19, no. 2, pp. 405–434, Aug. 2014.

[17] P. Huang and B. Yuan, “Mining Massive-Scale Spatiotemporal Trajectories in Parallel: A Survey,” in Trends and Applications in Knowledge Discovery and Data Mining, Springer, 2015, pp. 41–52.

[18] R. Moussalli, M. Srivatsa, and S. Asaad, “Fast and Flexible Conversion of Geohash Codes to and from Latitude/Longitude Coordinates,” in Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on, 2015, pp. 179–186.

[19] “no sql benchmark - Google Search.” [Online]. Available: https://www.google.rw/search?q=no+sql+benchmark&oq=no+sql+benchmark&aqs=chrome..69i57j0l5.4785j0j7&sourceid=chrome&ie=UTF-8. [Accessed: 22-Nov-2017].

[20] J. Han, E. Haihong, G. Le, and J. Du, “Survey on NoSQL database,” in Pervasive computing and applications (ICPCA), 2011 6th international conference on, 2011, pp. 363–366.

[21] “What is Apache Cassandra?,” Planet Cassandra, 18-Jun-2015. [Online]. Available:http://www.planetcassandra.org/what-is-apache-cassandra/. [Accessed: 23-Feb-2016].

[22] “CQL.” [Online]. Available: http://docs.datastax.com/en//cassandra/2.0/cassandra/cql.html. [Accessed: 23-Feb-2016].

[23] “Stratio/cassandra-lucene-index,” GitHub. [Online]. Available: https://github.com/Stratio/cassandra-lucene-index. [Accessed: 23-Mar-2016].

[24] “Apache CouchDB.” [Online].

Available: http://couchdb.apache.org/. [Accessed: 16-Aug-2016].

[25] "couchbase/geocouch," GitHub. [Online].

Available: https://github.com/couchbase/geocouch. [Accessed: 16-Aug-2016].

[26] tutorialspoint.com, "MongoDB Overview," www.tutorialspoint.com. [Online]. Available: http://www.tutorialspoint.com/mongodb/mongodb_overview.htm. [Accessed: 23-Feb-2016].

[27] "MongoDB for GIANT Ideas," MongoDB. [Online].

Available: https://www.mongodb.com/. [Accessed: 23-Feb-2016].

[28] MongoDB installation, "Sharding Introduction MongoDB Manual 3.2," https://github.com/mongodb/docs/blob/master/source/core/sharding-introduction.txt. [Online]. Available:https://docs.mongodb.org/manual/core/sharding-introduction/. [Accessed: 24-Feb-2016].

[29] J. Worsley and J. D. Drake, Practical PostgreSQL. O'Reilly Media, Inc., 2002.

[30] "Multi-node setup on Ubuntu or Debian — Citus 5.1.0 documentation." [Online]. Available: http://docs.citusdata.com/en/v5.1/installation/production_deb.html. [Accessed: 16-Aug-2016].

[31] "PostGIS — Spatial and Geographic Objects for PostgreSQL." [Online]. Available: http://postgis.net/. [Accessed: 16-Aug-2016].

[32] "RethinkDB: the open-source database for the realtime web." [Online]. Available: https://www.rethinkdb.com/. [Accessed: 16-Aug-2016].

[33] "Stratio/cassandra-lucene-index," GitHub. [Online]. Available: https://github.com/Stratio/cassandra-lucene-index. [Accessed: 30-Mar-2016].

[34] "Consistency & Cassandra," Planet Cassandra, 10-Apr-2013. [Online]. Available:http://www.planetcassandra.org/blog/consistency-cassandra/. [Accessed: 29-Sep-2016].

[35] "How To Create a Sharded Cluster in MongoDB Using an Ubuntu 12.04 VPS," DigitalOcean. [Online].

Available: https://www.digitalocean.com/community/tutorials/how-to-create-a-sharded-cluster-in-mongodb-using-an-ubuntu-12-04-vps. [Accessed: 29-Sep-2016].

[36] “Multi-node setup on Ubuntu or Debian — Citus 5.1.0 documentation.” [Online].
Available: http://docs.citusdata.com/en/v5.1/installation/production_deb.html. [Accessed: 16-Aug-2016

APPENDIX

I. Writing Throughput (operations per second) and latency in milliseconds

1. Dataset0

a. Sequential writing

Table 3. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	18222.02	0.55	5178.01	0.55	24437.98	0.72	39848.78	0.39	14714.92	0.41
2nodes	26006.40	0.54	6538.31	0.49	24437.98	0.72	39848.78	0.39	14714.92	0.41
3nodes	35899.21	0.42	9113.57	0.9	24437.98	0.72	39848.78	0.39	14714.92	0.41
4nodes	48311.46	0.51	15709.71	0.80	24437.98	0.72	39848.78	0.39	14714.92	0.41

Table 4. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.036	0.000045	0.044	0.000055	0.039	0.000052	0.033	0.000039	0.049	0.000041
2nodes	0.037	0.000051	0.043	0.000049	0.039	0.000052	0.033	0.000039	0.049	0.000041

3nodes	0.037	0.000044	0.045	0.00002	0.039	0.000052	0.033	0.000039	0.049	0.000041
4nodes	0.037	0.000050	0.045	0.000040	0.039	0.000052	0.033	0.000039	0.049	0.000041

b. Random writing

Table 5. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	15488.72	0.75	3883.51	0.65	20039.14	0.71	29886.59	0.81	9417.55	0.72
2nodes	22105.44	0.61	4903.73	0.65	20039.14	0.71	29886.59	0.81	9417.55	0.81
3nodes	30514.33	0.72	6835.18	0.62	20039.14	0.71	29886.59	0.81	9417.55	0.81
4nodes	41064.74	0.59	11782.28	0.58	20039.14	0.71	29886.59	0.81	9417.55	0.81

Table 6. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	Lat.	Stdv	Lat.	**Stdv**
1node	0.0383	0.000056	0.048	0.000066	0.043	0.000058	0.032	0.000052	0.054	0.000061
2nodes	0.0385	0.000062	0.046	0.000053	0.043	0.000058	0.032	0.000052	0.054	0.000061
3nodes	0.0392	0.000057	0.050	0.000059	0.043	0.000058	0.032	0.000052	0.054	0.000061
4nodes	0.0399	0.000060	0.0499	0.000064	0.043	0.000058	0.032	0.000052	0.054	0.000061

2. Dataset1

a. Sequential writing

Table 7 Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	16976.68	0.65	4799.31	0.75	22395.15	0.72	34522.88	0.83	10033.11	0.63
2nodes	23414.21	0.68	6426.20	0.68	22395.15	0.72	34522.88	0.83	10051.77	0.63
3nodes	35255.01	0.73	8919.24	0.61	22395.15	0.72	34522.88	0.83	10033.11	0.63
4nodes	47707.11	0.69	15081.17	0.53	22395.15	0.72	34522.88	0.83	10033.11	0.63

Table 8. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.046	0.000049	0.053	0.000059	0.049	0.000072	0.039	0.000078	0.057	0.000082
2nodes	0.046	0.000064	0.052	0.000061	0.049	0.000072	0.039	0.000078	0.057	0.000082
3nodes	0.047	0.000061	0.054	0.000065	0.049	0.000072	0.039	0.000078	0.057	0.000082
4nodes	0.047	0.000050	0.054	0.000062	0.049	0.000072	0.039	0.000078	0.057	0.000082

b. Random writing

Table 9. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	13241.81	0.76	3263.53	0.83	16796.36	0.81	23820.78	0.64	5518.21	0.71

2nodes	18263.09	0.73	4369.81	0.82	16796.36	0.81	23820.78	0.64	5528.47	0.71
3nodes	27498.91	0.62	6065.08	0.85	16796.36	0.81	23820.78	0.64	5518.21	0.71
4nodes	37211.55	0.69	10255.19	0.73	16796.36	0.81	23820.78	0.64	5518.21	0.71

Table 10. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.0463	0.000051	0.054	0.000061	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063
2nodes	0.0467	0.000054	0.053	0.000055	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063
3nodes	0.0467	0.000063	0.055	0.000071	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063
4nodes	0.0467	0.000058	0.055	0.000062	0.0483	0.000065	0.0455	0.000075	0.0573	0.000063

II. Reading Throughput (operations per second) and latency in milliseconds

1. Dataset0

A. Distance Query

a. Sequential Reading

Table 11. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3791460.81	0.76	1581316.58	0.67	3087332.38	0.65	1800943.88	0.86	2315499.28	0.68

2nodes	4265393.42	0.73	1852399.42	0.82	3601887.77	0.76	3241699	0.85	2818868.69	0.74
3nodes	4874735.33	0.62	2091418.70	0.81	4052123.75	0.89	4052123.75	0.73	3087332.38	0.66
4nodes	5358180.16	0.69	2493614.61	0.75	4987229.23	0.68	5186718.4	0.91	4052123.75	0.83

Table 12. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.032	0.000042	0.042	0.000051	0.033	0.000062	0.031	0.000074	0.041	0.000062
2nodes	0.033	0.000063	0.041	0.000062	0.031	0.000054	0.034	0.000065	0.039	0.000074
3nodes	0.035	0.000061	0.043	0.000058	0.031	0.000045	0.034	0.000056	0.041	0.000062
4nodes	0.036	0.000068	0.045	0.000079	0.031	0.000063	0.034	0.000058	0.042	0.000067

b. Random Reading

Table 13. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3033168.65	0.67	1233426.93	0.79	2346372.60	0.76	1332698.47	0.76	1713469.47	0.73
2nodes	3412314.73	0.89	1444871.55	0.75	2737434.71	0.83	2398857.26	0.83	2085962.83	0.82
3nodes	3899788.27	0.74	1631306.59	0.72	3079614.05	0.71	2998571.57	0.82	2284625.96	0.59
4nodes	4286544.13	0.79	1945019.4	0.86	3790294.21	0.59	3838171.61	0.82	2998571.57	0.81

Table 14. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**

1node	0.043	0.000053	0.058	0.000068	0.043	0.000072	0.042	0.000072	0.0515	0.000078
2nodes	0.048	0.000056	0.060	0.000068	0.045	0.000054	0.047	0.000073	0.052	0.000076
3nodes	0.048	0.000052	0.061	0.000061	0.045	0.000049	0.0475	0.000046	0.054	0.000064
4nodes	0.0482	0.000059	0.064	0.000062	0.045	0.000060	0.0475	0.000069	0.056	0.000072

B. K-nearest Query

a. Sequential Reading

Table 15. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3813763.52	0.79	2091418.70	0.84	3412314.73	0.75	2593359.2	0.63	2701415.83	0.71
2nodes	4630998.57	0.78	2493614.61	0.82	3813763.52	0.73	2818868.69	0.74	3087332.38	0.82
3nodes	5271055.28	0.83	2818868.69	0.91	4502359.72	0.75	3412314.73	0.66	3601887.77	0.64
4nodes	5737520.35	0.89	3412314.73	0.77	5105037.79	0.78	4322265.33	0.84	4052123.75	0.74

Table 16. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.031	0.000061	0.039	0.000072	0.031	0.000065	0.033	0.000076	0.037	0.000079
2nodes	0.032	0.000065	0.040	0.000056	0.032	0.000081	0.035	0.000091	0.038	0.000083
3nodes	0.034	0.000063	0.042	0.000057	0.032	0.000067	0.035	0.000077	0.041	0.000082
4nodes	0.037	0.000067	0.044	0.000077	0.032	0.000078	0.036	0.000079	0.043	0.000066

b. Random Reading

Table 17. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3089148.45	0.87	1652220.78	0.75	2661605.49	0.91	1996886.58	0.68	2026061.87	0.75
2nodes	3751108.84	0.86	1969955.54	0.72	2974735.55	0.83	2170528.89	0.76	2315499.28	0.72
3nodes	4269554.78	0.77	2226906.26	0.73	3511840.58	0.84	2627482.34	0.69	2701415.83	0.69
4nodes	4647391.48	0.75	2695728.64	0.76	3981929.48	0.85	3328144.30	0.81	3039092.81	0.82

Table 18. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	**Stdv**
1node	0.042	0.000071	0.044	0.000081	0.042	0.000076	0.043	0.000084	0.042	0.000095
2nodes	0.044	0.000067	0.0460	0.000074	0.0435	0.000074	0.044	0.000083	0.044	0.000072
3nodes	0.045	0.000087	0.0465	0.000081	0.0438	0.000081	0.044	0.000097	0.0455	0.000086
4nodes	0.045	0.000076	0.047	0.000067	0.0438	0.000067	0.044	0.000078	0.046	0.000069

C. Range Query

a. Sequential Reading

Table 19. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	4026955.27	0.91	2593359.2	0.94	3704798.85	0.87	3601887.77	0.91	3241699	0.93
2nodes	4874735.33	0.93	2946999.09	0.82	4502359.72	0.85	4322265.33	0.97	3769417.44	0.86
3nodes	5402831.66	0.71	3241699	0.84	4987229.23	0.79	4987229.23	0.84	4052123.75	0.79
4nodes	6483398	0.83	3813763.52	0.92	5402831.66	0.78	5893998.18	0.95	4987229.23	0.83

Table 20. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.029	0.000053	0.035	0.000061	0.0297	0.000059	0.030	0.000071	0.032	0.000072
2nodes	0.032	0.000071	0.034	0.000057	0.031	0.000057	0.030	0.000059	0.033	0.000057
3nodes	0.033	0.000056	0.037	0.000077	0.031	0.000058	0.032	0.000067	,0.034	0.000065
4nodes	0.033	0.000065	0.038	0.000065	0.031	0.000058	0.032	0.000066	0.035	0.000063

b. Random Reading

Table 21. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3181294.67	0.81	2022820.17	0.73	2815647.13	0.87	2665396.95	0.76	2334023.28	0.79
2nodes	3851040.91	0.83	2298659.29	0.71	3421793.38	0.86	3198476.34	0.79	2713980.55	0.65
3nodes	4268237.01	0.91	2528525.22	0.84	3790294.21	0.86	3690549.63	0.91	2917529.1	0.75
4nodes	5121884.42	0.93	2974735.55	0.85	4106152.06	0.75	4361558.65	0.69	3590805.04	0.78

Table 22. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv
1node	0.041	0.000063	0.045	0.000075	0.040	0.000065	0.042	0.000063	0.042	0.000071
2nodes	0.042	0.000068	0.0445	0.000056	0.041	0.000059	0.0425	0.000069	0.043	0.000063
3nodes	0.043	0.000064	0.046	0.000055	0.041	0.000057	0.0425	0.000067	0.044	0.000063
4nodes	0.043	0.000061	0.047	0.000067	0.041	0.000064	0.0425	0.000065	0.045	0.000054

D. Region Query

a. Sequential Reading

Table 23. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3813763.52	0.89	831204.87	0.93	2401258.51	0.85	1706157.36	0.79	1157749.64	0.98
2nodes	4322265.33	0.91	1137438.24	0.85	2946999.09	0.79	2401258.51	0.84	1271254.50	0.94
3nodes	4987229.23	0.79	1350707.91	0.88	3241699	0.87	3412314.73	0.78	1706157.36	0.87
4nodes	5637737.39	0.84	1800943.88	0.85	4052123.75	0.81	4322265.33	0.82	2235654.48	0.89

Table 24. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.030	0.000054	0.032	0.000061	0.029	0.000056	0.030	0.000062	0.031	0.000053
2nodes	0.031	0.000049	0.034	0.000062	0.030	0.000054	0.031	0.000053	0.033	0.000061
3nodes	0.031	0.000057	0.036	0.000054	0.030	0.000061	0.032	0.000058	0.034	0.000059

4nodes	0.031	0.000059	0.037	0.000056	0.030	0.000054	0.032	0.000062	0.036	0.000057

b. Random Reading

Table 25. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2974735.55	0.82	631715.70	0.76	1800943.88	0.78	1279618.02	0.81	856734.73	0.91
2nodes	3371366.96	0.83	864453.06	0.78	2210249.31	0.79	1800943.88	0.74	940728.33	0.87
3nodes	3890038.8	0.69	1026538.01	0.82	2431274.25	0.89	2559236.05	0.83	1262556.45	0.67
4nodes	4397435.16	0.78	1368717.35	0.67	3039092.81	0.81	3241699	0.76	1654384.31	0.71

Table 26. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	**Stdv**
1node	0.040	0.000049	0.044	0.000051	0.039	0.000045	0.039	0.000053	0.042	0.000054
2nodes	0.042	0.000061	0.0445	0.000054	0.0395	0.000057	0.0395	0.000062	0.043	0.000053
3nodes	0.042	0.000056	0.046	0.000061	0.039	0.000063	0.043	0.000053	0.045	0.000064
4nodes	0.042	0.000052	0.0475	0.000058	0.039	0.000059	0.043	0.000056	0.046	0.000061

2. Dataset1

A. Distance Query

a. Sequential Reading

Table 27. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3033168.65	0.76	1249240.10	0.82	2408119.25	0.89	1386726.79	0.92	1759779.45	0.78
2nodes	3412314.73	0.87	1463395.54	0.89	2809472.46	0.83	2496108.23	0.95	2142340.20	0.81
3nodes	3899788.27	0.78	1652220.78	0.91	3160656.525	0.88	3120135.28	0.89	2346372.60	0.92
4nodes	4286544.13	0.67	1969955.54	0.77	3890038.8	0.85	3993773.16	0.76	3079614.05	0.71

Table 28. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.041	0.000065	0.052	0.000055	0.043	0.000058	0.038	0.000064	0.049	0.000058
2nodes	0.042	0.000059	0.051	0.000058	0.040	0.000061	0.041	0.000056	0.048	0.000062
3nodes	0.043	0.000061	0.053	0.000053	0.040	0.000059	0.041	0.000061	0.049	0.000056
4nodes	0.045	0.000057	0.055	0.000051	0.040	0.000062	0.041	0.000051	0.050	0.000054

b. Random Reading

Table 29. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2365871.55	0.84	949738.74	0.84	1783243.18	0.79	999523.85	0.86	1267967.40	0.91
2nodes	2661605.49	0.76	1112551.09	0.76	2080450.38	0.81	1799142.94	0.75	1543612.49	0.92
3nodes	3041834.85	0.73	1256106.07	0.73	2340506.67	0.76	2248928.68	0.79	1690623.21	0.87

4nodes	3343504.42	0.71	1497664.93	0.71	2880623.60	0.82	2878628.71	0.84	2218942.96	0.89

Table 30. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.054	0.000053	0.069	0.000055	0.0535	0.000057	0.053	0.000065	0.0615	0.000051
2nodes	0.059	0.000068	0.071	0.000059	0.057	0.000069	0.056	0.000064	0.063	0.000055
3nodes	0.061	0.000052	0.073	0.000051	0.057	0.000056	0.0582	0.000061	0.064	0.000054
4nodes	0.0632	0.000065	0.075	0.000049	0.057	0.000052	0.0583	0.000057	0.067	0.000062

B. K-nearest Query

a. Sequential Reading

Table 31. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2898460.28	0.91	1610392.40	0.87	2661605.49	0.96	2048753.76	0.65	2161132.66	0.77
2nodes	3519558.91	0.87	1920083.25	0.75	2974735.55	0.84	2226906.26	0.69	2469865.90	0.78
3nodes	4006002.01	0.83	2170528.89	0.76	3511840.58	0.87	2695728.64	0.73	2881510.22	0.85
4nodes	4360515.46	0.76	2627482.34	0.68	3981929.48	0.89	3414589.61	0.67	3241699	0.73

Table 32. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**	**CouchDB**	**MongoDB**	**PostgreSQL**	**RethinkDB**

	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.041	0.000054	0.049	0.000059	0.041	0.000063	0.043	0.000051	0.047	0.000056
2nodes	0.042	0.000061	0.050	0.000067	0.042	0.000058	0.045	0.000065	0.048	0.000068
3nodes	0.044	0.000059	0.052	0.000053	0.042	0.000062	0.045	0.000062	0.049	0.000053
4nodes	0.047	0.000057	0.054	0.000066	0.042	0.000056	0.046	0.000055	0.051	0.000052

b. Random Reading

Table 33. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2285969.85	0.86	1239165.58	0.77	2022820.176	0.73	1537602.66	0.83	1580328.26	0.86
2nodes	2775820.54	0.76	1477466.65	0.85	2260799.02	0.85	1671307.24	0.75	1806089.44	0.85
3nodes	3159470.53	0.72	1670179.70	0.83	2668998.84	0.87	2023161.40	0.83	2107104.35	0.76
4nodes	3439069.70	0.75	2021796.48	0.86	3026266.40	0.85	2562671.11	0.82	2370492.39	0.75

Table 34. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	**Stdv**
1node	0.053	0.000051	0.057	0.000063	0.051	0.000053	0.052	0.000054	0.052	0.000052
2nodes	0.055	0.000049	0.0580	0.000062	0.054	0.000071	0.055	0.000064	0.056	0.000051
3nodes	0.057	0.000061	0.060	0.000054	0.054	0.000059	0.055	0.000069	0.059	0.000068
4nodes	0.057	0.000053	0.062	0.000052	0.054	0.000067	0.055	0.000058	0.060	0.000058

C. Range Query

a. Sequential Reading

Table 35. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	3221564.22	0.78	2048753.76	0.81	2889743.10	0.84	2773453.58	0.79	2463691.24	0.86
2nodes	3899788.27	0.87	2328129.28	0.76	3511840.58	0.75	3328144.30	0.74	2864757.25	0.82
3nodes	4322265.33	0.75	2560942.21	0.69	3890038.8	0.70	3840166.50	0.73	3079614.05	0.79
4nodes	5186718.4	0.75	3012873.18	0.71	4214208.7	0.82	4538378.6	0.85	3790294.21	0.76

Table 36. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**
1node	0.039	0.000059	0.045	0.000052	0.039	0.000054	0.040	0.000051	0.042	0.000056
2nodes	0.042	0.000052	0.044	0.000065	0.041	0.000053	0.040	0.000061	0.043	0.000059
3nodes	0.043	0.000057	0.047	0.000051	0.041	0.000061	0.042	0.000054	0.044	0.000065
4nodes	0.043	0.000048	0.048	0.000055	0.041	0.000053	0.042	0.000071	0.045	0.000062

b. Random Reading

Table 37. Throughput (Th) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2481409.84	0.79	1496886.93	0.74	2139891.81	0.76	1999047.71	0.74	1727177.22	0.76
2nodes	3003811.91	0.69	1701007.87	0.82	2600562.97	0.69	2398857.26	0.83	2008345.61	0.82

3nodes	3329224.87	0.72	1871108.66	0.65	2880623.60	0.81	2767912.22	0.85	2158971.53	0.72
4nodes	3995069.84	0.81	2201304.30	0.59	3120675.57	0.66	3271168.99	0.92	2657195.73	0.87

Table 38. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	**Stdv**
1node	0.050	0.000056	0.055	0.000056	0.049	0.000054	0.052	0.000049	0.051	0.000051
2nodes	0.052	0.000062	0.0545	0.000062	0.051	0.000063	0.053	0.000056	0.054	0.000062
3nodes	0.054	0.000053	0.056	0.000059	0.051	0.000067	0.053	0.000059	0.055	0.000064
4nodes	0.054	0.000049	0.057	0.000063	0.051	0.000054	0.053	0.000062	0.056	0.000054

D. Region Query

a. Sequential Reading

Table 39. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2898460.28	0.86	640027.75	0.85	1872981.64	0.77	1347864.32	0.56	926199.71	0.88
2nodes	3284921.65	0.59	875827.44	0.73	2298659.29	0.86	1896994.22	0.74	1017003.60	0.79
3nodes	3790294.21	0.73	1040045.09	0.77	2528525.22	0.73	2695728.64	0.77	1364925.89	0.83
4nodes	4284680.41	0.69	1386726.79	0.64	3160656.52	0.75	3414589.61	0.88	1788523.58	0.75

Table 40. Latency (Lat.) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Lat.	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**	**Lat.**	**Stdv**

1node	0.041	0.000046	0.043	0.000063	0.04	0.000056	0.041	0.000062	0.042	0.000064
2nodes	0.042	0.000051	0.045	0.000052	0.041	0.000057	0.042	0.000052	0.044	0.000059
3nodes	0.042	0.000063	0.047	0.000054	0.041	0.000063	0.043	0.000056	0.045	0.000061
4nodes	0.042	0.000059	0.048	0.000062	0.041	0.000051	0.043	0.000059	0.047	0.000047

b. Random Reading

Table 41. Throughput (Th) and its standard deviation (Stdv)

Nodes	**Cassandra**		**CouchDB**		**MongoDB**		**PostgreSQL**		**RethinkDB**	
	Th.	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**	**Th.**	**Stdv**
1node	2320293.73	0.75	486421.09	0.86	1368717.35	0.75	959713.51	0.59	633983.70	0.96
2nodes	2629666.22	0.69	665628.86	0.45	1679789.48	0.67	1350707.91	0.69	696138.96	0.84
3nodes	3034230.26	0.67	790434.27	0.84	1847768.43	0.73	1919427.03	0.87	934291.77	0.95
4nodes	3429999.42	0.78	1053912.36	0.76	2309710.53	0.72	2431274.25	0.83	1224244.39	0.81

Table 42. Latency (Lat.) and its standard deviation (Stdv)

Nodes	Cassandra		CouchDB		MongoDB		PostgreSQL		RethinkDB	
	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	Stdv	Lat.	**Stdv**
1node	0.052	0.000053	0.054	0.000056	0.051	0.000063	0.051	0.000055	0.053	0.000056
2nodes	0.053	0.000061	0.0545	0.000054	0.052	0.000053	0.051	0.000047	0.054	0.000064
3nodes	0.053	0.000054	0.056	0.000049	0.052	0.000058	0.054	0.000051	0.055	0.000057
4nodes	0.053	0.000051	0.0575	0.000063	0.052	0.000053	0.054	0.000056	0.056	0.000046

Chapter Three

Section 1

Published as C. Niyizamwiyitira, L. Lundberg, and M. Svahnberg, “Evaluation of Voice-driven Web Application Architecture,” in Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on, 2012, pp. 555–562.

Evaluation of Voice-driven Web Application Architecture

Abstract

This paper quantifies the implications and trade-offs of three different architectures for voice driven web application, architectures are implemented as prototypes. The prototypes differ from each other by either using recording, or Text To Speech (TTS) as server based, or TTS as client based to process output speech. A typical application used in this paper, is the most dynamic weather information source which is presented as web feeds or Really Simple Syndication (RSS) feeds. The evaluated quality attributes are performance, maintainability, and development effort. The empirical results show that, each system's architecture has a different quality profile, for instance, one architecture has the lowest development time but the highest maintainability cost, and another has the lowest bandwidth requirements but the highest development cost. Finally, suggestions about optimal choice of system architecture according to the quality requirements of the final system are drawn.

Keywords

Voice based web, IVR application, Web voice quality attributes, Voice driven web evaluation, voice driven architecture.

3.1.1 Introduction

In the last decade, there has been a rapid development in Voice-over-IP and extensible mark-up language (XML) technologies to provide voice based services. These sophisticated technologies lead the Interactive Voice Response (IVR) to become increasingly a common IT solution in successful enterprises [1]. People with low computer literacy or who have no internet accessibility can use their mobile phones to access information. In fact, approximately 23 percent of the world population only, have access to WWW [2]; voice driven web such as IVR applications and worldwide telecom web (WWTW) were developed to fill a gap of the rest 77 percent that most of them have access to a telephone.

The main deployment architecture of voice driven web is composed of three main parts; the first part is the core part which is voice application server, it holds the voice applications. The second part is the speech engine server that hosts text to speech (TTS) and automatic speech recognition (ASR). The third is the voice gateway or voice browser that performs communication tasks between the user and voice server over session initiative protocol (SIP) or telephone [3–5].

In voice driven web architecture, play recordings instead of using text to speech is more preferable, because of the quality of the voice that is natural when using recording. However, it can be extremely expensive and require a great deal of effort to record and appropriately concatenate audio files to play whenever this information is required [6]. In this paper, we evaluate the performance, maintainability, and development effort quality attributes with regard to different architectures according to the processing of output speech. In the first architecture, information recorded is played back to the user.

In the second, TTS processed on the server side, reads the information prompt and then audio is sent to the user, and in the third, the information is transmitted as text based that will be read by TTS on the end user. This evaluation is conducted through experiment with respect to 3 prototypes that represent above mentioned architectures. All three prototypes provide the weather information for Blekinge cities in Sweden.

The rest of this paper is organized as follows;

The second section presents background, problem, motivations, implications of solving the problem, and research question. The third section shows how the research has been done and clarifies the research methodology. The fourth section presents results and analysis based on the refined research questions. The fifth Section discusses results and their contributions, and threats validity. The sixth section draws conclusion and recommendations.

3.1.2 Background

3.1.2.1 Voice driven web

Voice driven web is a Voice XML based application that provides automated interaction condition for callers to retrieve information from web through telephone keypad or speech recognition [1]. It controls and responds to callers by utilizing speech technologies. In fact, impressive growth of the World Wide Web and new special customers' needs to access information on the web, required new services and fast development, hence, voice driven web applications have been risen [6].
Currently, we are familiar with WWW and many people worldwide frequently utilize various services provided over the internet. However, there is a large number of people mostly in developing country who do not have access to the internet and hence do not benefit from web services [1].
To cope with the above mentioned, alternatives are; IVR that provide automatic and interactive call, spoken web known as World Wide Telecom Web is one of the newest voice driven web technology (from 2005) that was introduced by IBM. It aims to make accessible voice driven web services for untouched population whom most of them have access to a landline or cell phone. In addition, the spoken web is different from others IVR existing approaches, in the sense that it simplifies voice application development to the extent of becoming usable by non IT savvy [2]. In voice driven web, speech to text plays an important role to render information into speech, additionally, the implementation of rendering speech impacts on bandwidth consumption for voice driven web. Therefore, this paper presents the evaluation of voice driven web on architectural level with focus on three different ways of rendering speech, either using recordings or TTS on the server side or TTS implemented on the end user device.

3.1.2.2 Software Quality attributes and their evaluation techniques

Nowadays, one of the major issues in information system development is the quality attributes. Rather than designing and implementing the correct functionality in products, the main challenge is to satisfy the quality requirements; such as performance, maintainability, reliability, flexibility and usability [7]. The main aim of utilizing voice driven web is to provide favorable condition for users to access the information from web in an easy way through voice medium. In this paper, quality attributes are evaluated on the level of the system architecture. In [8], system architecture is defined as a part of the software development process that delineates the quality attributes of a system and the environment.

The most relevant quality attributes in the context of this paper are performance, maintainability, and development effort.

The research question that guides this paper is as follows;

How do architectures impact on the voice driven web quality with regards to performance, maintainability, and development effort quality attributes?

This paper introduces a comparative study between three different architectures with regards to speech rendering techniques. Specifically, we consider 3 major quality attributes that are performance, maintainability, and development effort. To reach this goal, it is necessary to use proper measurements function as explained below;

Performance is the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, and cost [9], [10].

In this paper, the performance is measured with focus on the cost per call based on the payload size because we believe that the price paid to get voice service is a highly considerable resource.

Maintainability is the ease with which a software or component can be modified to correct faults, improve performance or other attributes, or adapt to change environment [11]. IEEE and ISO/IEC 14764 added that in a common view, maintenance is all about fixing bugs, they also proposed software correction through analyze system fault, information and conduct root cause analysis. Maintainability attribute has been found to be a critical attribute to meet, as this averagely comprises 60 percent of software total cost [12].

Architecture based maintenance prediction method is a method to predict software system maintainability in the early stage of development based on its architecture [7]. The method defines a maintenance profile which is a set of change scenarios representing perfective and adaptive maintenance tasks. Perfective is enhancements to software which provide additional functionality or enhance existing functionality whereas, adaptive is adapting the software to changes in the environment, and changing the software to support future maintenance or operation [13].
Using this method, other maintenance categories such as corrective are left behind because they are too abstract to be relevant at architectural design stage [14]. This method differs from other scenario based methods such as Scenario-Based Architecture Analysis (SAAM), in that, it does not involve all stakeholders, hence require less resource and time, instead it provides an instrument to the software architects that allows them to repeatedly evaluate architecture design. As all stakeholders are not involved in the assessment at the stage of this work, we use architecture based maintenance prediction method.
The development effort is used for a number of purposes; such as tradeoff and risk analysis, software improvement, investment analysis, and time in hours that has been taken for development. Originally developed by Howard Rubin in the late 1970s as Quest (Quick Estimation System), it was subsequently integrated into the Management and Computer Services (MACS) line of products as ESTIMACS [Rubin 1983]. This method has about 5 sub estimation methods; among those we choose system development effort estimation, this model estimates development effort as total effort hours [15]. We choose to estimate development effort in terms of hours like how much time can you spend to complete a given task, because it is a format that can be used in agile projects and by clients, and which leads to more optimistic effort estimates compared with the traditional request that is like how much effort will it take to complete a given task[16].

In this paper, we focus on time that the development has taken as the total effort while assuming that the developer is familiar with the system development.

3.1.2.3 Related Work

Bernhard Suhm in [17], conducted IVR usability engineering using guidelines and analyses of end to end calls, the author only considered the usability quality attribute. In [18], Akhil Mittal proposes different ideas that can be followed to do manual testing and quality monitoring of IVR. The author gives idea of what to consider while choosing between Manual Testing and Automation Testing to test IVR applications, why automation is difficult in testing IVR applications and use of automation tools in monitoring IVR applications. In [19] and [20], authors also worked on usability assessment of IVR. In [21], Hyeong et al worked on design and implementation of real time news services using RSS and voiceXML, thereafter they evaluated users' satisfaction by comparing typical RSS service scenario and voiceXML service scenario. M bargein et al. studied architecture for voice enabled interface over local wireless networks[22], where they propose a distributed architecture that split TTS and ASR into two parts each of both. They propose that Natural Language Processing (NLP) and grammar generator of TTS and ASR respectively to be implemented on the server, whereas the speech synthesis and speech recognition could be processed in the terminal. In this paper, we measure 3 different quality attributes of voice driven web application at architectural level; those are performance in terms of cost per call, maintainability, and development effort.

3.1.3 Methodology

Experiment method is used to conduct this research; it is defined as research method that is based on observed and measured phenomena. Experiments are usually performed in development, evaluation, and problem solving research [23].
The author continues to explain that, experiments are referred to as research-in-the-small, since they are concerned with a limited scope and most often are run in a laboratory setting. They are often highly controlled and hence also occasionally referred to as controlled experiment. An example of a controlled experiment in software engineering is, to compare different methods for inspections. Montgomery, Siegel, Castellan, Robson (as cited by Wohlin et al.) in their publication, said that for this type of study, methods for statistical

inference areapplied with the purpose of showing with statistical significance, that one method is better than the other [17]. Hence, as we are evaluating three different architectures, experiment method fits in to get the empirical comparative results.
Three different architectures that are implemented as prototypes, are subsequently detailed; the platform that is used is Voxeo Prophecy [24] for experiment purpose, this platform offers free documented version that most IVR and voice driven web application do not offer. We use the platform on premises or locally hosted, additionally, we host the voice application on voxeo developer portal from where we assign a local telephone number to our voice driven application. In the end, we let users try the system; Figure 1 shows the system use case for providing weather information; the user calls in, the system replies with a greeting message and options to get the specific region weather information. User has two input options to select the location;
either input digits on the phone keypad or saying the city name and then the system replies accordingly.

3.1.3.1 Prototype 1

Prototype 1 is shown on Figure 2; it represents the architecture that uses prerecording to render the information, the output is the recording that is played back. ASR and DTMF tone recognizer, help for speech recognition of the input.
The weather information is recorded in the following format (PCM, Mono, and 8 KHz) using audacity 2.0.1, the standard audio format for telephone. The recordings are stored in audio repository, when a call comes in; it is received by voice browser that transfers the request to vxml document that acts as voice user interface which retrieve the recording from audio repository.

3.1.3.2 Prototype 2

Prototype 2, as shown on Figure3 is the architecture that consists of TTS as well as automatic speech recognition (ASR) implemented on the server side. When a call comes in, the request is recognized either by ASR or DTMF according to the input options respectively. The request is received by voice browser that transfers the request to vxml document[1] that acts as voice user

[1] Vxml document term is used interchangeably with voice user interface as well as voice application.

interface which retrieves the correct weather information web feeds[2] page with the help of DOM parser, this is a JavaScript RSS parser. The vxml document interacts with the web without the intervention of logic side server. The web feeds are transferred to vxml documents as prompts and then they are read by TTS. Hence the output audio is sent to end user.

3.1.3.3 Prototype 3

Prototype 3, as shown on Figure4, is the architecture that consists of TTS as well as automatic speech recognition (ASR). However, TTS is installed on the end user device instead of server side, as it is in the prototype 2. As it is shown on the Figure4, Media Resource Control Protocol (MRCP) is embedded in the body of Session Initiation Protocol (SIP) for the purpose of controlling Text to Speech [25] that is held in the terminal. The weather information is presented as text prompt from web feeds as the same process in prototype 2, thereafter the prompts are sent in the form of text format that will be read by TTS on end user device; hence only text data will be sent over network.

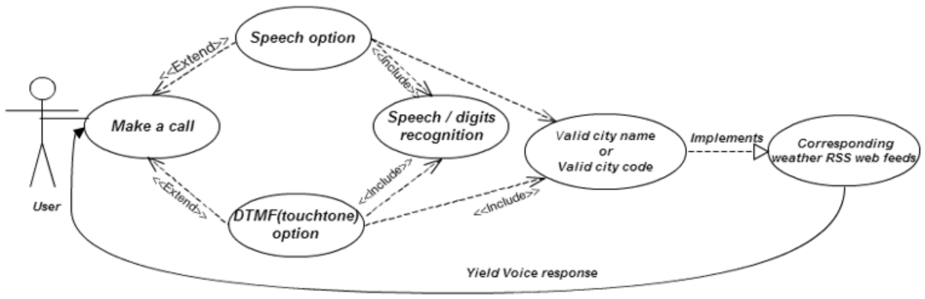


Figure 1. System Functionalities.

[2] Web feeds term is used interchangeably with RSS feeds.

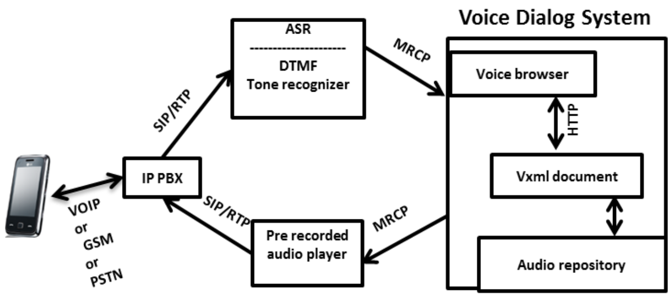


Figure 2. Architecture where playing record is used.

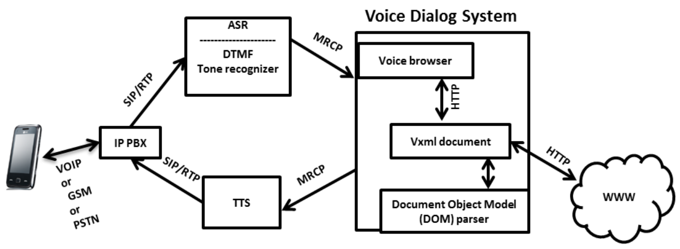


Figure 3. Architecture where TTS/ASR implemented on the server side.

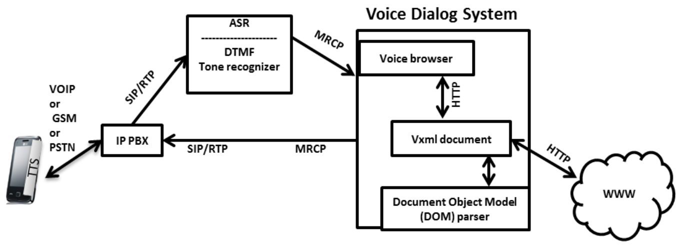


Figure 4. Architecture where TTS implemented on the user side.

In all 3 different prototypes, user inputs either voice or dual tone multi frequency (DTMF) tone on the telephone keypad. Session Initiative Protocol (SIP) is used to setup the session between user and the voice browser. Real Time Transfer Protocol (RTP) is used to exchange input and output audio stream between the end user and ASR/ TTS server. Media Resource Control Protocol (MRCP) controls the transmission between the voice browser and speech server.

3.1.3.4 Performance Evaluation

The performance is evaluated in terms of the cost per call it is calculated in (1).

$$cost = \frac{data}{payload} * \frac{currency}{time} = \frac{kb}{kb/_{min}} * \frac{\$}{min} \quad (1)$$

Where *data* is the amount of total data to be transmitted, to average this data; for prototype 1 we measure the size of the recordings. For prototype 2, we take the text prompt that would be spoken, and then we use the TTS to read it, thereafter the output size of audio is the data to be transmitted. Whereas, prototype 3, we consider the size of data in the text format, to be transmitted. *Payload* is the amount of data traffic per unit of time; 64 kbps is the bandwidth payload when using G 711. PCM codec [26], [27] the standard for telephone codec that has the highest mean opinion score (MOS).
$/min, is the currency over time or the unit price per min.
Kb is data unit, simply kilobit unit.

3.1.3.5 Maintainability Evaluation

To evaluate the maintainability as mentioned in the section 2, prediction method based on the architecture is used. Using the maintenance profile, the architecture is evaluated using so-called scenario scripting and the expected maintenance effort for each change scenario is also evaluated. Based on this data, the required maintenance effort for a software system can be estimated [7]. Table 1 shows the scenario changes. As prototype 1 plays the prerecorded audio, we consider also the time taken to record the update as the additional maintainability effort. To estimate this time, we assume that the reader has the average speech reading of 125 words per minute as indicated in [28], that is the average speed for a comprehensive speech.
http://www.yr.no/place/Sweden/Blekinge/Karlskrona/hour_by_hour_detailed.html is one of the cases of the source webpages for a single city, updates of web feeds or RSS of next 48 hours are done every hour with every hour that consists of 27 words, and it means 27*48 that equals to 1296 words to be recorded every hour. The estimated time of recording every hour is 1296/125 that equals to 10.3 minutes; if we consider three cities used in our case, this time will be 31 minutes needed for every hour.

If we consider a period of one year of 8765 hours, the updating time is given by K in (2);

$$k = \frac{8765\ hrs}{60\ min} * 31\ min = 4529hrs \qquad (2)$$

Table 1 shows the maintenance profile i.e. a set of change scenarios representing perfective and adaptive maintenance tasks, description and its category. It also shows the weight of each scenario; the weight is assigned by domain expert or by software architect and it represents the occurring probability of the corresponding scenario. The total weight probability of all scenarios equals to one.

Table 1. Maintenance Scenario Profile

Scenario	**Category**	**Scenario description**	**Weight**
1	Usability friendliness	Input modalities (change from DTMF to speech).	0.25
2	Location forecast	Add new city.	0.45
3	Information Source	Change information Source, i.e. form one website to another. (Connection between voice user interface and www).	0.10
4	Forecast period	Change forecast period from 5 day to 2 day or to 1 weather forecast.	0.20
Total			1.0

Table 2. Estimated Component Size

Component	**Size (LOC)**
DTMF Grammar	15 LOC
ASR Grammar	15 LOC
Voice application	150 LOC
RSS feed Parser	20 LOC

Table 3. Impact Analysis per Scenario

Scenario # =>affected prototype#	**Dirty Component**	**Volume (LOC)**
S1 =>P1,2,3	DTMF grammar (100%) + ASR Grammar (100%)	(15+15)=30
S2=>P1	Voice application (5%) +DTMF Grammar (100%) + ASR Grammar (100%)	(150*0.05)+15+ 15=37.5
S2 =>P2,3	Voice application (5%)+DTMF Grammar + ASR Grammar +RSS feed Parser	(150*0.05)+15+ 15+20 =57.5
S3 =>P1	Voice application (20 %)	150*0.2=30
S3 =>P2,3	Voice application (5 %) + RSS feed Parser (100%)	(150*0.05)+20 =27.5
S4 =>P1	Voice application (20 %)	150*0.2=30
S4 =>P2,3	RSS feed Parser (100%)	20

Table 2 shows estimated system component size, the size given is only for three cities used; system components size is measured in lines of code (LOC) and it is said to be estimated because it is a prototype, i.e. incomplete version of the system. The prototypes are coded by one person using Extensible Markup Language (XML 1.0) for grammar and for encoding documents in a format that is both human and machine readable, Voice Extensible Markup Language (VXML 2.1) for specifying interactive voice dialogues between a human and a computer, and JavaScript for grabbing data from web. The web source is written in XML 1.0 with RSS 2.0, it can be found on; http://www.yr.no/place/Sweden/Blekinge/Karlskrona/rss.xml.
Table 3 presents the scenario scripting, i.e. change impact analysis; it is done by investigating the required changes to the components of the application architecture [29]. The percentage number indicates how much the component is affected by a scenario. For example, scenario 3 (S3) affects prototypes 2 and 3 with change of about 5 % of the voice application volume and 100% of RSS

feed parser volume. Finally, the average maintenance is calculated in (3) using the results obtained from Table 1, 2, and 3.

$$M_eP_n = \sum_{1}^{k_S}\left[P\left(S_n\right) * \sum_{m=1}^{kc} V\left(S_n, C_m\right) \right]. \tag{3}$$

Where $P\left(S_n\right)$ is the probability weight of a scenario n

k_s : Number of scenarios

n : Scenario number

k_C : Number of components in architecture

$V\left(S_n, C_m\right)$: Volume of affected component m in scenario *n*

M_eP_n : Maintainability effort for prototype P_n

3.1.3.6 Development effort Evaluation

In this paper, we focus on development effort in terms of time that the development has taken. The time has been measured using a time tracker tool called baralga 1.7, this tool allows to keep track of the time spent working on different prototypes. It has the option of setting different tasks that compose a prototype; using the above mentioned tool, we recorded the time while working on each part, specifically design, coding, and testing of prototype development.

3.1.4 Results and Analysis

3.1.4.1 Performance

Using (1), we only calculate the cost for the outbound data, inbound data which is about information request; is same for all prototypes hence it does not make any difference between all prototypes. Let's consider the cost of one single call case for each of the prototypes. Suppose that the price is 0.22$ per minute per data transfer in real time, either voice data or text data as it is usually for VoIP charges. With G.711 that takes 3840 kb/min, prototypes 1, 2, 3 outbound data size equals to 10813 kb, 6904 kb, and 5.8 kb respectively for a single call as explained in section 3.1.3.4 above.

Prototype 1:

$$cost = \frac{10813.44\ kb}{3840\ {kb}/{min}} * \frac{0.22\$}{min} = 0.62\$ \tag{4}$$

Prototype2:

$$cost = \frac{6904\ kb}{3840\ {kb}/{min}} * \frac{0.22\$}{min} = 0.395\$ \tag{5}$$

Prototype 3:

$$cost = \frac{5.8\ kb}{3840\ {kb}/{min}} * \frac{0.22\$}{min} = 0.00033\$ \tag{6}$$

3.1.4.2 Maintainability

The predicted maintainability effort is calculated using the (3), we assume that the maintenance has a perfective median productivity, and this is when the latter is reported as 1.7 LOC/day [13] as cited by different authors [7], [30–32]. 1.7 LOC/ day are equivalent to 0.2 LOC per hour assuming that we work 8 hours per day. Maintenance equation for prototype 1 is given below in (7);

$$\begin{aligned} prototype1 &= (0.25 * 30) + (0.45 * 37.5) + (0.10 * 30) + (0.20 * 30) \\ &= 33.375\ LOC\ per\ change \end{aligned} \tag{7}$$

Suppose we have 10 maintenance tasks for a predicted period of one year;

$$M = 33.375\ LOC * 10 = 333.75 LOC \tag{8}$$

M in (8) is the total lines of code to be maintained for one year period.
Finally, the maintainability of prototype 1 in terms of time for one year period is equivalent to T as given in (9);

$$T = \frac{333.75\ LOC}{0.2 LOC/hr} = 1669 hrs \tag{9}$$

Additionally as mentioned in (2), the recording time will be an added maintaining time.

The maintainability effort for prototype 2 is similar to prototype 3 and as given in (10).

$$rototype\ 2\ and\ 3 = (0.25 * 30) + (0.45 * 57.5) + (0.10 * 27.5) + (0.20 * 20)$$

$$= 40.125\ LOC\ per\ change \tag{10}$$

Suppose we have 10 maintenance tasks for a predicted period of one year, 10 changes are given by M below;

$$M = 40.125\ LOC * 10 = 401.25\ LOC \tag{11}$$

And the final maintainability effort in terms of time T is;

$$T = \frac{401.25\ LOC}{0.2\ LOC/_{hr}} = 2006.25\ hrs \tag{12}$$

3.1.5 Development Effort

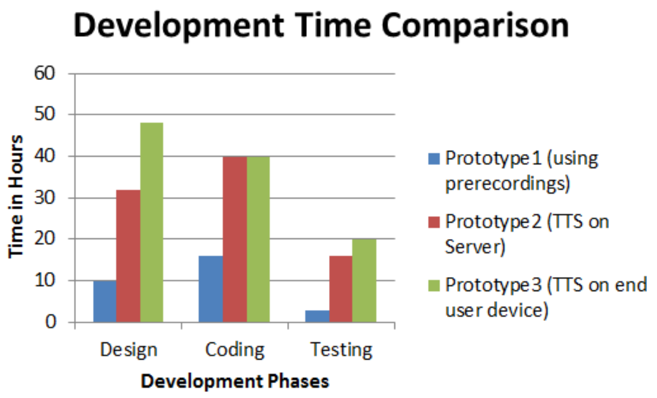


Figure 5. Development Time Comparison.

Table 4. Results Summary

Quality Factor	**Prototype 1**	**Prototype 2**	**Prototype 3**
Performance (in terms of cost)	0.62$	0. 395$	0.00033$
Maintainability	1669 hrs. [Extra recording time: 4526 hrs.]	2006 hrs.	2004 hrs.
Development Time	29 hrs.	88hrs	108 hrs.

Table 4 shows the results summary of the evaluation; the results are based on the architecture, i.e. these results will have the same trends on different applications that use similar architecture. Moreover, the values in the table 4 will increasingly or decreasingly change according to how big the application is; and if the same architecture is used, the ratio of quality factor values will hold as the example we have explained in this paper e.g. the application could be weather information, football news, etc.

3.1.5 Discussions

In terms of cost effectiveness, the performance attribute results as shown on Table 4, is evaluated for 3 different prototypes. The cost for inbound data is same for all prototypes, to mean that only outbound can make the difference. Prototype 3 is the cheapest due to transmit text data which is always of small size comparing to voice data. Though prototype 2 transmits voice data as well as prototype 1, prototype 2 outperforms prototype 1 because TTS audio output is faster than the recordings due to nature of human voice, prerecorded speech are bigger in size.

The costs given are just for one case; they may go higher or lower according to the data size, but always with the same proportional trends as the above explained case.

The maintainability effort prediction as shown on Table 4 shows the adaptive and perfective of the prototypes. Prototype 1 has the lowest coding time, nevertheless, it is the hardest to adapt, because recordings must be done at each update; that is very hard and maybe impossible for very dynamic and live system such as RSS that gets updated very frequently, hence effective for static information system. In our case, the updates are done every hour hence, we get additional time for adaptive weather forecast system as show in (2); prototype 1 approximately requires three times effort needed for prototypes 2 or 3.

In the same perspective of lines of code (LOC);

Prototypes 2 and 3 have the same effort because both, they gets updated information from http://www.yr.no/place/Sweden/Blekinge. For testing, prototype 2 is accessed on + 46 455 68 06 05 where hosted on voxeo server. Prototype 1 is also accessed on + 46 455 68 06 02.

The development time as plotted on Figure 5 and summarized in Table 4 shows that, prototype 1has the least time for development, this is due to coding time that is shorter comparing to other two prototypes. Whereas for prototype 2 and 3, extra coding time is spent on coding the speech recognition grammar, text to speech prompt, DOM parser and voice user interface which is more complex in prototype 2 and 3 than prototype 1.

To accurately use proper measurements of quality attributes; we use the measurements that have been recommended or used by a wide range of researchers with similar or approximately same results. The development time given on Figure 5 shows only the time that design, coding and testing have taken. In this paper, the prototype 3 that implements TTS on the end user device, is limited to end user device that support SIP, RTP and MRCP protocols, it means computer, smart client such as PDA, smart phone, in our case we used computer as the end user device. Moreover, to avoid speaking speed and accuracy issue, prerecording was done by an English native speaker person.

3.1.6 Conclusions

This paper presented the evaluation of voice driven web on the level of its architecture and with regards to performance, maintainability, and development effort quality attributes. The results show that the output speech rendering impact on the above quality attributes differently. Depending on the requirements of the system, prototype 1 can be used for example, in a system that does not need to get updated very often and requires a natural human voice, however, the cost per call is higher than the other prototypes. When the system requires cheap cost but limited to smart end users' device then prototype 3 is the right choice. And when the system requires to operating on all types of telephone; prototype 2 is preferred.
The optimal choice of system architecture will depend on the quality requirements of the final system. Our study quantified the implications and trade-offs of three different architectures for a voice driven web application; hence the later can help the system architects when developing voice driven applications. To enhance this research, the coming work is to apply voice driven web in different field as we are heading ubiquitous network society. Additionally, improve prototype 3 by reducing the needs of speech technologies such as power consumption and memory constraints which are critical in mobile telephone. Moreover, usability attribute will be evaluated in future work. The drawbacks of maintainability evaluation method used in this paper, is to get accurate maintenance profile, to alleviate this, scenario should be given likelihood weight by different stakeholders. Development effort estimation method also may be influenced by the skills of the developer as well as the resources such as equipment, etc. In order to get a much closer result to reality, many developers would code the same work in parallel while tracking the time used by each and at the end, the average time will be

calculated. This method will be very costly; however, the results could be more accurate.

References

[1] D. Amyot and R. Simoes, “Combining VoiceXML with CCXML - A comparative study,” in 2007 4th Annual IEEE Consumer Communications and Networking Conference, CCNC 2007, January 11, 2007 - January 13, 2007, 2007, pp. 342–346.

[2] A. Kumar, S. K. Agarwal, and P. Manwani, “The spoken web application framework - User generated content and service creation through low-end mobiles,” in International Cross Disciplinary Conference on Web Accessibility, W4A 2010, April 26, 2010 - April 27, 2010, 2010, p. ACM’s Special Interest Group on Accessible Computing; The Mozilla Foundation; Zakon Group; Microsoft; IBM.

[3] S. K. Agarwal, A. Jain, A. Kumar, and N. Rajput, “The World Wide Telecom Web browser,” in 1st ACM Symposium on Computing for Development, DEV 2010, December 17, 2010 - December 18, 2010, 2010.

[4] D. Prylipko, D. Schnelle-Walka, S. Lord, and A. Wendemuth, “Zanzibar OpenIVR: An open-source framework for development of spoken dialog systems,” in 14th International Conference on Text, Speech and Dialogue, TSD 2011, September 1, 2011 - September 5, 2011, 2011, vol. 6836 LNAI, pp. 372–379.

[5] A. A. Atayero, C. K. Ayo, I.-O. Nicholas, and A. Ambrose, “Implementation of ‘ASR4CRM’: An automated speech-enabled customer care service system,” in IEEE EUROCON 2009, EUROCON 2009, May 18, 2009 - May 23, 2009, 2009, pp. 1712–1715.

[6] J. R. Lewis, P. M. Commarford, and C. Kotan, “Web-based comparison of two styles of auditory presentation: All tts versus rapidly mixed tts and recordings,” in 50th Annual Meeting of the Human Factors and Ergonomics Society, HFES 2006, October 16, 2006 - October 20, 2006, 2006, pp. 723–727.

[7] P. Bengtsson and J. Bosch, “Architecture level prediction of software maintenance,” in Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on, 1999, pp. 139–147.

[8] L. Bass, P. Clements, and R. Kazman, Software architecture in practice. Addison-Wesley Professional, 2003.

[9] A. Neely, M. Gregory, and K. Platts, “Performance measurement system design: a literature review and research agenda,” International Journal of Operations & Production Management, vol. 15, no. 4, pp. 80–116, 1995.

[10] H. Y. Zhang, J. Homer, G. Einicke, K. Kubik, and others, “Performance comparison and analysis of voice communication over ad hoc network,” in Proceedings of the 1st Australian Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 06), 2006.

[11] J. Radatz, A. Geraci, and F. Katki, “IEEE standard glossary of software engineering terminology,” IEEE Std, vol. 610121990, p. 121990, 1990.

[12] R. L. Glass, “Frequently forgotten fundamental facts about software engineering,” Software, IEEE, vol. 18, no. 3, pp. 112–111, 2001.

[13] J. E. Henry and J. P. Cain, “A quantitative comparison of perfective and corrective software maintenance,” Journal of Software Maintenance: Research and Practice, vol. 9, no. 5, pp. 281–97, Oct. 1997.

[14] S. Anwar, M. Ramzan, A. Rauf, M. A. Jaffar, and A. A. Shahid, “A novel approach for architecture based software maintenance prediction,” International Journal of Innovative Computing, Information & Control, vol. 7, no. 6, pp. 3193–208, Jun. 2011.

[15] B. Boehm, C. Abts, and S. Chulani, “Software development cost estimation approaches - a survey,” Annals of Software Engineering, vol. 10, pp. 177–205, 2000.

[16] M. Jørgensen and S. Grimstad, “Software Development Effort Estimation—Demystifying and Improving Expert Estimation,” Simula Research Laboratory, pp. 381–403, 2010.

[17] B. Suhm, “IVR Usability Engineering using Guidelines and Analyses of end-to-end calls,” Human factors and voice interactive systems, pp. 1–41, 2008.

[18] A. Mittal, “Manual Testing and Quality Monitoring of Interactive Voice Response (IVR) applications,” International Journal of Computer Applications, vol. 4, no. 6, pp. 30–36, Jul. 2010.

[19] T. Ndwe, E. Barnard, R. Koen, and B. McAlister, “Efficiency measurements in IVR systems for oral users: consequences of differences in educational levels,” in Proceedings of the South African Institute of Computer Scientists and Information Technologists Conference on Knowledge, Innovation and Leadership in a Diverse, Multidisciplinary Environment, 2011, pp. 171–176.

[20] M. Greeff, L. Coetzee, and M. Pistorius, “Usability evaluation of the South African National Accessibility Portal interactive voice response system,” in

Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, 2008, pp. 76–85.

[21] H. J. Kwon, J. H. Shin, and K. S. Hong, “Design and implementation of enhanced real time news service using RSS and VoiceXML,” Human Interface and the Management of Information. Methods, Techniques and Tools in Information Design, pp. 677–686, 2007.

[22] M. Bagein, O. Pietquin, C. Ris, and G. Wilfart, “An Architecture for Voice-Enabled Interfaces over Local Wireless Networks,” in Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), Orlando,(USA, FL), 2003.

[23] C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, and A. Wesslén, “Experimentation in software engineering: an introduction,” 2000.

[24] “VoiceXML and CCXML Developer Site.”, www.evolution.voxeo.com.

[25] D. R. <drb@relay.prime.com>, “Using Media Resource Control Protocol over SIP.” [Online]. Available: http://tools.ietf.org/html/draft-robinson-mrcp-sip-00. [Accessed: 16-Oct-2012].

[26] J. Walker and J. Hicks, “Planning for VoIP,” NetIQ Corporation white paper, December, 2002.

[27] A. H. Rabassa, “Simulation Platform for the Planning and Design of Networks Carrying VoIP Traffic,” Carleton University, 2010.

[28] D. C. Rubin, “51 properties of 125 words: A unit analysis of verbal behavior,” Journal of Verbal Learning and Verbal Behavior, vol. 19, no. 6, pp. 736–755, 1980.

[29] K. H. Bennett and V. T. Rajlich, “Software maintenance and evolution: a roadmap,” in Proceedings of the Conference on the Future of Software Engineering, 2000, pp. 73–87.

[30] J. Li, T. St\aalhane, J. M. W. Kristiansen, and R. Conradi, “Cost drivers of software corrective maintenance: An empirical study in two companies,” in Software Maintenance (ICSM), 2010 IEEE International Conference on, 2010, pp. 1–8.

[31] T. Imai, Y. Kataoka, and T. Fukaya, “Evaluating software maintenance cost using functional redundancy metrics,” in Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International, 2002, pp. 299–306.

[32] B. Roy and T. C. N. Graham, “Methods for evaluating software architecture: A survey,” School of Computing TR, vol. 545, p. 82, 2008.

Chapter Three

Section 2

Published as C. Niyizamwiyitira and L. Lundberg, “Performance evaluation and prediction of open source speech engine on multicore processors,” in Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems, 2013, pp. 345–352.

Performance evaluation and prediction of open source speech engine on multicore processors

Abstract

This paper quantifies the performance of the core part of voice driven web using free and open source speech engine; the speech engine which is very high computation demanding, it consists of Automatic Speech Recognition (ASR) and Text To Speech (TTS). Two open source programs, Sphinx-4 and FreeTTS-1.2.2 are used for ASR and TTS respectively. These two programs are executed on 2 different hardware multicore processors with 4 hyperthreaded cores, and 8 cores respectively. The response time with respect to the load variance and the number of cores is measured and predicted using a linear regression model.
The results show that, the response time is linear with respect to the input length, this property can be used to directly predict the response for any input length. Moreover, though the response time and the speed up increases as the number of cores increases, the regression coefficients and number of threads reveal that ASR benefits from multicore. The speedup factor for ASR is 1.56 for 8 cores. However for FreeTTS, though being sequential the speed up from the program itself is insignificant, there is about 1. 43 speedup for 8 cores, that comes from the system's contribution. Our findings show that the generalization of the results for multicore processor does not apply to hyperthreading. This paper presents the investigation that is useful for educators, researchers, and applications' developer in voice based applications 'domain.

Keywords— Speech recognition, Text to speech, Voice driven web, Multicore performance, Performance prediction, linear regression, Open Source.

3.2.1 Introduction

Nowadays voice based-application are emerging very fast, there exists many voice-based application from different vendors that are not open to researchers. Therefore it becomes difficult to know what is behind scene. For that reason, we investigate free and open source voice engines in order to better understand voice application's phenomenon. Generally, there are three main architectures for Voice driven web; the first architecture uses prerecording to render the information, and the server plays back the recordings. The interaction between server and terminal is done through Dual Tone Multi Frequency (DTMF) and/or ASR (Automatic Speech Recognition). This is a user friendly approach since one uses recordings from humans. However, this architecture is not suitable when the information is updated frequently due to almost impossible and tedious recording work [1].

The second architecture consists of ASR as well as TTS (Text To Speech) that are implemented on the server side. Where, the server performs all speech processing; this server-based processing permits fast processing and consequently fast response time. This architecture is scalable because it simplifies the introduction of new components such as hardware resources, adding new services, and applying load balancing thus putting fewer requirements on the terminals

The third architecture performs all of the ASR and TTS processing on the terminal instead of server side and the traffic between the server and the terminal are only text messages over the network instead of voice as it is for the previous architectures. This architecture is limited to smart phones that can handle ASR and TTS, and it requires installations of ASR and TTS on the terminal. ASR and TTS are by nature computational demanding; hence when they are implemented on the handheld device the performance decreases due to limited computational capability of the latter. The architecture tradeoffs show that the architecture that implements ASR and TTS on the server side has better performance [2].

Voice driven web should provide real-time communication and handle many client requests simultaneously. Therefore, it requires huge processing resources. For this reason, the second architecture which is server-based speech processing has been found promising if it is implemented in an appropriate server environment.

The speech engine is highly computational demanding, and multicore processors are necessary. As far as this paper is concerned, the performance of speech engine is studied where ASR and TTS are executed on multicore processors with shared memory. In this paper, the performance metric is the response time of ASR and TTS. We will measure the response time on two different multicore processors. Based on these measurements we will build a performance prediction model. The predicted response time as a function of the input load length is derived according to linear regression model; the response time as a function of the number of cores is also investigated. Voice driven web like any other voice application can be developed using open source ASR and TTS. In this paper, we use free and open source, Sphinx-4 and FreeTTS-1.2.2. The rest of the paper is structured as follows; the second section presents background, motivations, implications of this study, and related work. The third section clarifies the research methodology. The fourth section presents results. The fifth section discusses results and their contributions; and study validity. The sixth section draws conclusion and provides recommendations.

3.2.2 Background

3.2.2.1 Concept Overview

Voice driven web is a Voice XML based application that provides automated interaction condition for callers to retrieve information from web through telephone keypad or speech recognition. The architecture of voice driven web has three main parts; (1) a voice browser that interact directly with the terminal, (2) a speech technology server, i.e., ASR that recognize the input speech from the end-user and TTS that reads the content of the web in response to the end-user request, and (3) the back end which is composed of a web proxy and the World Wide Web (www). ASR and TTS are the most computationally demanding parts in voice driven web. Hence, implementing this part on powerful processing hardware would improve the overall response time which is a crucial factor for the quality of voice driven web systems.

In 2008, there existed two and four cores machines, some experts believe that by 2017 embedded processors could support 4,096 cores, server CPUs might have 512 cores and desktop chips could use 128 cores [5].To select the appropriate hardware architecture, performance evaluation of different candidates is vital. In this paper, the performance of ASR/TTS on multicore processor architectures is evaluated. We also attempt to predict the performance for a large number of cores. The performance is evaluated as the response time. The response time for ASR is the time taken to recognize a certain audio input until we get the text output, whereas for TTS, it is the time taken to get audio output from a certain text input.

In voice driven web, the response time is influenced by network/ communication delays, and speech processing delays. In this paper, we focus on evaluating speech processing. If we assume that the network is good enough to carry the voice, the response time of the voice driven web can be estimated based on the ASR and TTS response times. We use open source speech recognition named Sphinx-4 [6]. It is a state-of-the-art continuous-speech, and speaker-independent systems based on hidden Markov (HMM) decoded using the frame synchronous Viterbi algorithm [7]. It has also N-gram language models. Sphinx-4 is distributed under an academic BSD-style license. The code and binaries are free for commercial and non-commercial use. We also use open source text to speech named FreeTTS. It is based on CMU's Flite, which is derived from the Festival and the FestVox project, from Carnegie Mellon University [3]. FreeTTS is also released under a BSD license. The ideas behind choosing Sphinx-4 and FreeTTS to conduct this study are; (1) portability because they are portable entirely written in Java and are flexible, adjustable to the extent that they can be edited and configured to fit the user requirements, (2) the software is free and easily available. The investigation of the speech engine as open source on multicore processor helps the researchers and developers to adapt to new hardware technology.

3.2.2.2 Related work

In [8], the authors studied fine-grained application concurrency in a HMM-based inference engine for large vocabulary continuous speech recognition (LVCSR). They measured the performance based on Intel core i-7 and GTX280 NVIDIA processors. Sphinx-4 that also uses HMM and Viterbi algorithm is studied.

In [9], Yoshizawa et al. proposed a scalable architecture for real-time speech recognizers based on word hidden Markov models (HMMs). This architecture architecture effectively uses parallel computations on the word HMM structure.

A performance model for multicore environments has been studied in [10] where Chen et al. presented an online technique for estimating the performance and the power consumption of interacting processes in multi-programmed and multicore environments. Willie et al. compared the performance of FreeTTS which is implemented in Java, and C. They used a dual core processor for their experimentation. The authors found that FreeTTS that is developed in Java performs better than in C in terms of fast response time and algorithm modifications becomes easier [11]. The same authors studied the performance of Sphinx-4, they explained the framework and properties of Sphinx-4, and they presented preliminary performance measures on a dual core processor [6]. In [12], Adve came up with a simple deterministic model for parallel program performance prediction, using deterministic values to represent mean task times including communication. This model can be used to quantify and understand program performance, and to predict the impact of system and program changes. In [13], the authors studied how to predict the performance of multicore on Solaris containers, using linear regression. They come up with optimistic results where the correlation between response time and the number of CPUs and number of threads is linear in logarithmic space.

Across literature, the performance prediction of ASR and TTS for multicore processor with high number of cores has not been studied. Sphinx-4 and FreeTTS were particularly studied on dual core processor. Whereas in this paper, we study the performance of those on dual quad core and Intel core i-7. We provide performance evaluation of the speech engine, i.e., ASR and TTS. We measure the response time, the speedup, the CPU utilization, and number of threads. The measurement is conducted on different number of cores are done on different hardware. Moreover, the response time as a function of input load is measured and prediction is derived based on linear regression.

3.2.3 Methodology

3.2.3.1 Experiment setup

In the experiment, we use CMU Sphinx-4 automatic speech recognition, which is an open source speech recognizer developed by Carnegie Melon

University (CMU) [14]. Sphinx-4 has two packages, the bin which is enough for creating application, and the source which is modifiable and containing the test part to fix bugs after modifying the code. The packages are downloadable under sourceforge repository, statistics shows that 3285 downloads are done until 23rd of August 2013.

We also use FreeTTS 1.2.2 which is also an open source text to speech which is built on CMU's flite, FreeTTS has three downloadable packages which are bin, source and test [4] [12]. The bin package is sufficient for creating application, but if you like to modify code then source is the correct package to use. If it happens to make change to code, test package will help to fix bugs and regression that might introduced during changes. Downloads under source forge repository show that 438 downloads have been done until 26th April, 2013. The statistics for both FreeTTS and Sphinx-4 show how much these open sources are popular in the speech application domain.

The following equipment is used in this experiment: (1) Intel Core i-7-2620M / 2.7 GHz, 8GB RAM, dual-core with hyperthreading of two threads per core, i.e., 4 logical cores. It runs the Ubuntu operating system 12.10 LTS, 32 bits. (2) Intel(R) Xeon(R) E5335 @ 2.00GHz dual quad core, 16 GB RAM, with 8 cores without hyperthreading, which runs Ubuntu 12.10 LTS, 64 bits.

We isolated the ASR/TTS servers and measured the response time for different loads. The CPU utilization and the number of threads are also measured. In this experiment, the input to the ASR is a continuous audio file with fifteen spoken digits during 8 seconds; the utterances are separated by silences [15]. This audio has similar characteristics as a typical speech from a user; this audio is recorded in a native speaker voice. The text that is used in this experiment, is the first five chapters of Alice's Adventures in Wonderland by Lewis Carroll [16]. It is a text of 70 minutes. We have chosen this text, because it was first used by the pioneer of FreeTTS for testing.

The initial length for FreeTTS is the first five chapters of Alice Adventure, it is about 10900 words. For a matter of varying the length of input load, we linearly concatenated up to 9 times the initial. The highest load is 10900*9= 98100 words.

The initial audio load for ASR is 80 seconds audio length. We vary the load by concatenating initial load until 9 times, i.e., the highest load is 80*9= 720 seconds. At every increase of load we run it 100 times in order to measure the average response time.

We use the Jvisualvm profiling tool for examining CPUs utilization, memory consumption and thread of the application [17].

3.2.3.2 ASR and TTS architecture

Figure 1 shows ASR recognition that is used in the experiment.

ASR main parts are feature extractor, hidden Markov and Viterbi decoding. Viterbi decoding is the most computation demanding part. The decoding uses the frame synchronous Viterbi algorithm, which is described in Figure 2. An example of this decoding is described in Figure 3, where the word “one” is decoded.
In Figure 3, for each state with more than one connection, Viterbi scores the state with the probability of the most probable connection. Then it keeps a backpointer to which state it is connected to. For example at time t=3, 0.06 is selected as the maximum probability the rest is pruned away. The pruning process follow beamwidth threshold, and the garbage collector helps in that, i.e., the pruner removes the terminal token and identifies the token and any unshared tokens that are unused. Then the garbage collector reclaims the associated memory [6]. Threads are used to score each state probability at each time frame. Each core executes a thread. The size of the thread pool for scoring states is set dynamically according to the number of available cores.
The processing of TTS is done in sequence with the following main steps; text normalization, linguistic analysis, lexical analysis, prosody generation, speech synthesis [11].

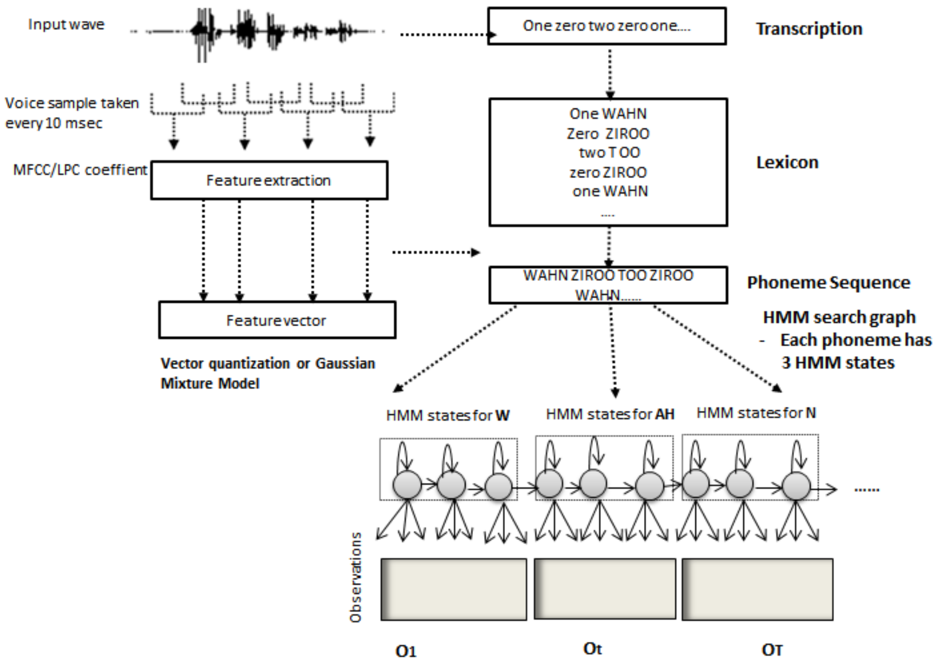


Figure 1. Architecture of Speech recognition processing steps.

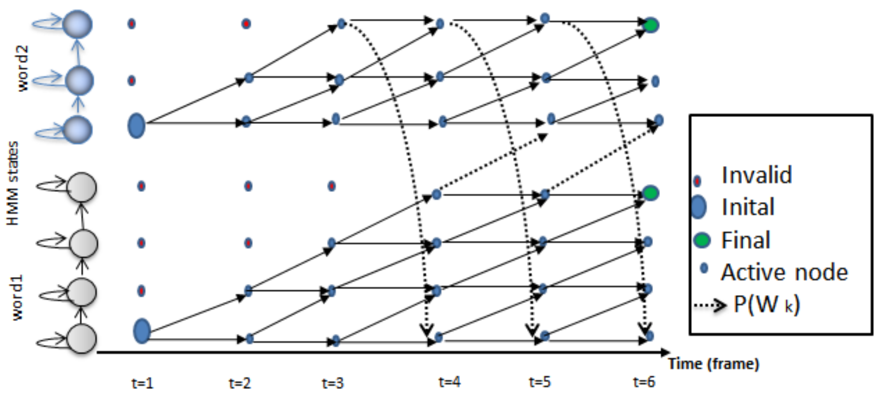


Figure 2. Connected word Viterbi Search.

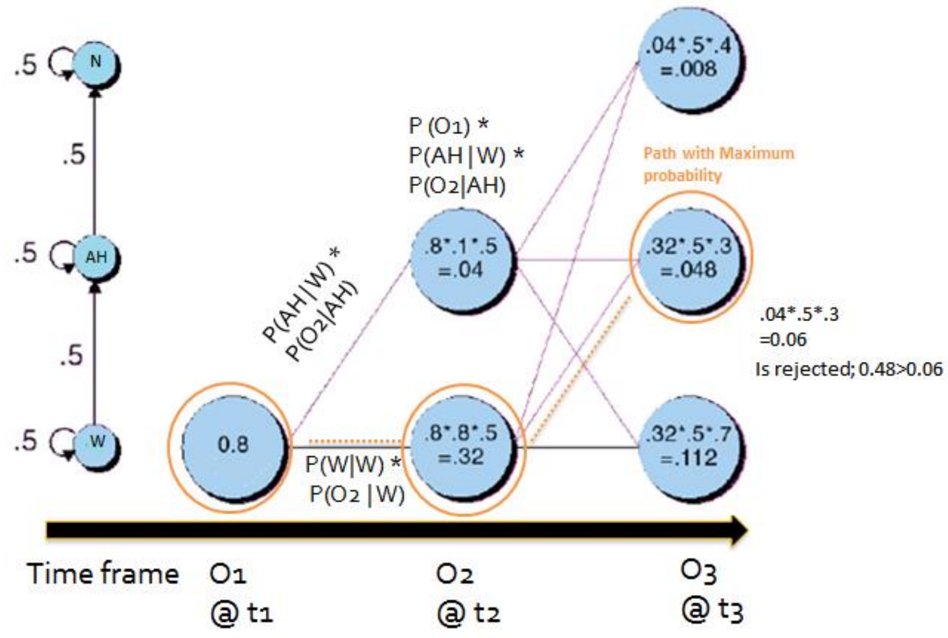


Figure 3. Example of Frame synchronous Viterbi decoding for a word "One".

3.2.3.3 Linear regression Performance Modeling

As described in the previous section, the maximum number of cores considered in the experiment is 8 cores. Based on the experimental results, we want to predict the performance of ASR/TTS for higher input load and higher number of cores. To do that, we have to apply a fitting function to the experimental results and extend the pattern for further input value. The linear regression is a well-known simple fitting function [18]. Therefore we use linear regression to extrapolate the experimental results. Linear regression fits a straight line through data to determine the slope and intercept with the *y-axis*. Using the linear regression method we get the least square regression line (LSRL) that is given in Equation 1 [19]. In the context of this paper, LSRL helps us to predict the response time with respect to input length.

$$f_i(x) = a_i + b_i x \tag{1}$$

In this case, *f(x)* is a dependent variable that represents the response time that is predicted based on the independent variable *x* which represents the input length; *a* is a regression constant and *b* is the regression coefficient and it is also the average change in the dependent variable for a unit change in the independent variable. The index *i* represents the number of cores.

Let $f(x) = \hat{y}$, then Equation 1 becomes Equation 2.

$$\hat{y} = a_i + b_i x \tag{2}$$

$$a_i = \frac{\Sigma[(x_i - \bar{x})(y_i - \bar{y})]}{\Sigma[(x_i - \bar{x})^2]} \tag{3}$$

$$b_i = \bar{y} - (a_i * \bar{x}) \tag{4}$$

Here, y_i is the measured response time on *i* cores.

After finding the regression equation, the coefficient of determination that helps to check how well this equation fits the data is given in Equation 5. The coefficient of determination R^2 ranges from 0 to 1; as its value increases as the dependent variable can be better predictable from the independent variable. $R^2 = 1$ represents perfect correlation between the variables in question, while $R^2 = 0$ represents no correlation [19].

$$R^2 = \{(1/n) * \Sigma[(x_i - \bar{x})(y_i - \bar{y})] / (\sigma_x * \sigma_y)\}^2 \tag{5}$$

Besides R^2, the Mean Squared Error (MSE) is also used to measure the precision between measured and predicted values based on regression method. When the measured and predicted values are more strongly correlated, MSE tends to approach zero.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{6}$$

Where *n* is the number of cores, $\hat{y}$ is the predicted response time.

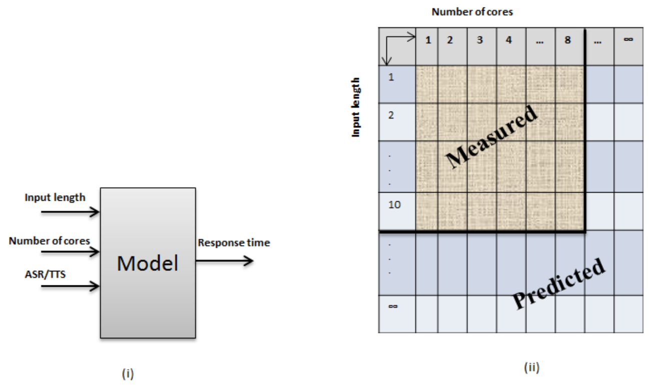


Figure 4. Performance prediction model.

3.2.4 Results

Figure 4 (i) shows an overview of the model; the inputs are different load length, number of cores, and the ASR/TTS case, the output from the model is the predicted response time. Equation 2 represents the output; and it is composed of two components *a* and *bx* that contribute to the response time and the speed. The response time is predicted in two dimensions. Figure 4 (ii) shows a matrix which would contain the response time in two dimensions; vertically, we have the response time as a function of the input length, and horizontally we have the response time as a function of the number of cores. Scenario 1 is the experiment on Intel xeon dual quadcore. Tables 1 and 2 show the regression parameters values for ASR and TTS respectively. Figures 5(i, ii) show the response time for ASR and TTS, it is the time that it takes to get the audio output from a text. The response time for ASR, is the time that it takes to get the text as output of an audio file. The figures show also the linear regression fitting line with respect to measured data; these lines are the predicted response times. Scenario 2 uses the same procedure as Scenario 1 on Intel core i-7. Tables 3 and 4 show the regression parameters' values. Figures 5(iii, iv) show the response time and the linear regression fitting line for ASR and TTS respectively. The regressions parameters' values are shown in Tables 1, 2, 3, and 4.

Table 1. Regression parameter’s values for ASR on 8 CPU

	1C	**2C**	**3C**	**4C**	**5C**	**6C**	**7C**	**8C**
A	7.77	5.56	4.93	4.80	4.71	4.68	4.50	4.50
B	1.28	1.18	1.10	0.97	1.06	1.08	0.92	0.83
R^2	0.98	0.99	0.98	0.99	0.98	0.98	0.99	0.99
MSE	0.10	0.02	0.02	0.01	0.10	0.10	0.11	0.10

Table 2. Regression parameter’s values for TTS on 8 CPU

	1C	**2C**	**3C**	**4C**	**5C**	**6C**	**7C**	**8C**
A	4.29	2.48	1.68	1.68	1.49	1.38	1.42	1.33
B	4.54	3.62	3.47	3.47	3.39	3.40	3.38	3.33
R2	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
MSE	0.14	0.17	0.09	0.11	0.09	0.17	0.19	0.17

Table 3. Regression of TTS on the 4 cores CPU

	1C	**2C**	**3C**	**4C**
A	3.37	1.14	1.76	1.16
B	3.53	2.82	2.81	2.80
R2	0.99	0.99	0.99	0.99
MSE	0.14	0.11	0.32	0.21

Table 4. Regression on ASR on the 4 cores CPU

	1C	**2C**	**3C**	**4C**
A	3.65	3.62	2.35	2.35
B	0.63	0.71	1.76	1.66
R2	0.96	0.98	0.95	0.95
MSE	0.07	0.02	0.01	0.01

Figure 6 and 7 are the results from JvisualVM for the ASR and TTS respectively. Figure 6 shows an example of typical ASR parallel scoring threads when 4 cores are enabled. We can see that the self-time of the concurrent method, i.e., java concurrent .Future Task which is composed of 4 parallel ThreadedAcousticScorer threads consume 100% of the CPU time that is used to complete the total work. Figure 7 shows a TTS example of threads' repartition according to CPU time consumption. The main thread handles all the work in a sequential manner. Under hot spot method, we see that, the IdentityHashMap method takes the highest percentage of CPU time.

Initially, when we run on a single core, we have 11 threads in TTS, 10 of those are for java virtual machine (JVM) functionality, i.e., they are not part of TTS program. Attach Listener, Thread-1 for audio output, Service Thread, C2 CompilerThread1, C2 CompilerThread0, Signal Dispatcher, Finalizer, Reference Handler, VM thread, VM periodic thread. Those threads do not consume significant CPU time. Another thread is the main thread and it is the actual thread of TTS.

It sequentially executes the program code. As the number of cores increase, we get additional threads that run in parallel; those are garbage collectors which are inherently parallel. Garbage collector contributes very little to the speed up of the program. The total threads number follows the Equation 7.

$$x_i = x_{i=1} + (\# cpus) \qquad (7)$$

Where *x* is the total number of thread and *i* is CPUs number
Initially, ASR runs on a single core with 9 threads for java virtual machine (JVM) functionality; Attach Listener, Service Thread , C2 CompilerThread1, C2 CompilerThread0, Signal Dispatcher, Finalizer, Reference Handler, VM thread, VM periodic thread. and the main thread that execute the whole functions of ASR. The main thread executes sequentially for a single core.

However, as the number of cores increases; the total number of threads increases according to Equation 8. There is one scoring thread increase and one thread garbage collector increase per one core increase. For example; on 4 cores we have 10+8=18 threads. Whereas on 8 cores we have 10+16=26.

$$x_i = x_{i=1} + 2(\# cpus) \qquad (8)$$

Where *x* is total number of thread and *i* is number of CPUs.
Figure 6 shows an example of typical parallel scoring threads when 4 cores

are enabled, how many times the threads have been invoked during the runtime. If we take a look in table 3 and 4, the “b” values, the relative speed up of the response time on Intel core i-7 is not following any trends, hence not easy to predict with respect to number of cores.

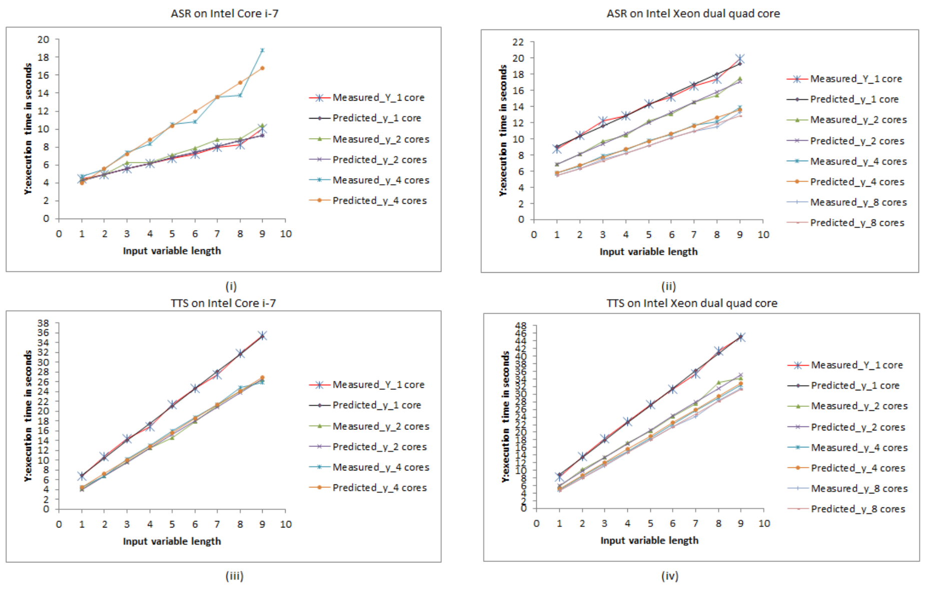
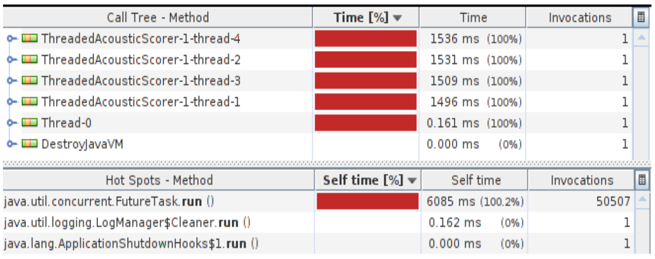


Figure 5. Measured and predicted response time for ASR and TTS.

Therefore we consider the relative speed up measured on Intel Xeon dual quad core where some conclusions can be drawn. The speed up measurements shows trends that could be predicted. In Table 6 and 7, at every input length, the relative speed up is normalized in a sense that the response time for one enabled core is said to be 1. This response time is divided by the response time for each more enabled cores. Values in Table 6 and 7 are given by equation 9. The response time for one core is expected to be the highest and the one for the 8 cores is expected to the smallest.

$$\frac{(\mathrm{Li},1\mathrm{C})}{(Li,\mathrm{iC})} \tag{9}$$

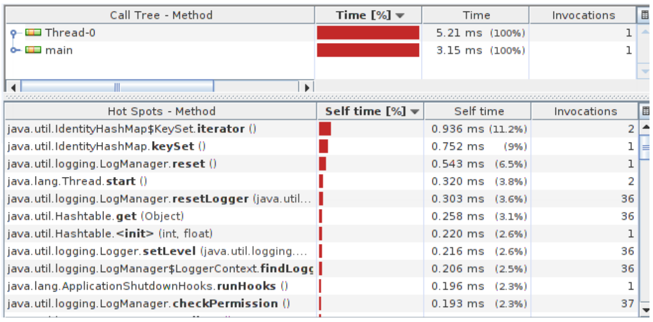
iC is number of enabled cores; Li is input length. Table 5 shows the number of threads for ASR and TTS with respect to number of cores. The CPU utilization on Intel-xeon is given in Figure 8. As a matter of representation, the CPU utilization shown in Figure 8 is for the smallest and for the biggest input length. For both ASR and TTS, the CPU utilization does not change significantly for different input load length on the same number of cores. Instead it decreases as the number cores increases (see Figure 8).



Call Tree - Method	Time [%]	Time	Invocations
ThreadedAcousticScorer-1-thread-4	🟥	1536 ms (100%)	1
ThreadedAcousticScorer-1-thread-2	🟥	1531 ms (100%)	1
ThreadedAcousticScorer-1-thread-3	🟥	1509 ms (100%)	1
ThreadedAcousticScorer-1-thread-1	🟥	1496 ms (100%)	1
Thread-0	🟥	0.161 ms (100%)	1
DestroyJavaVM		0.000 ms (0%)	1

Hot Spots - Method	Self time [%]	Self time	Invocations
java.util.concurrent.FutureTask.**run** ()	🟥	6085 ms (100.2%)	50507
java.util.logging.LogManager$Cleaner.**run** ()		0.162 ms (0%)	1
java.lang.ApplicationShutdownHooks$1.**run** ()		0.000 ms (0%)	1

Figure 6. ASR threads.



Call Tree - Method	Time [%]	Time	Invocations
Thread-0	🟥	5.21 ms (100%)	1
main	🟥	3.15 ms (100%)	1

Hot Spots - Method	Self time [%]	Self time	Invocations
java.util.IdentityHashMap$KeySet.**iterator** ()	🟥	0.936 ms (11.2%)	2
java.util.IdentityHashMap.**keySet** ()	🟥	0.752 ms (9%)	1
java.util.logging.LogManager.**reset** ()	🟥	0.543 ms (6.5%)	1
java.lang.Thread.**start** ()	🟥	0.320 ms (3.8%)	2
java.util.logging.LogManager.**resetLogger** (java.util...	🟥	0.303 ms (3.6%)	36
java.util.Hashtable.**get** (Object)	🟥	0.258 ms (3.1%)	36
java.util.Hashtable.**<init>** (int, float)	🟥	0.220 ms (2.6%)	1
java.util.logging.Logger.**setLevel** (java.util.logging....	🟥	0.216 ms (2.6%)	36
java.util.logging.LogManager$LoggerContext.**findLogg**	🟥	0.206 ms (2.5%)	36
java.lang.ApplicationShutdownHooks.**runHooks** ()	🟥	0.196 ms (2.3%)	1
java.util.logging.LogManager.**checkPermission** ()	🟥	0.193 ms (2.3%)	37

Figure 7. TTS threads.

Table 5. Number of Threads

#cores= >	1	2	3	4	5	6	7	8
ASR	10	14	16	18	20	22	24	26
TTS	11	13	14	15	16	17	18	19

3.2.5 Discussions

In ASR, the scoring of Viterbi algorithm is done in parallel, where at each time frame each HMM state is scored by a thread. The threads are generated with respect to the number of enabled cores (see Table 5). Moreover, the garbage collector which is important during pruning where it reclaims the memory at each path pruned out is also processed in parallel with all available cores. In TTS, all steps are processed in sequence. However, the garbage collector that helps to manage the memory is processed in parallel with all available cores. This garbage collector contributes almost 2 % to the total work. Figures 5 (i, ii, iii, iv) show that the response time for both ASR and TTS is linear as a function of the input length.

Table 6. ASR Normalized speedup

	1C	**2C**	**3C**	**4C**	**5C**	**6C**	**7C**	**8C**
L1	1.00	1.26	1.39	1.50	1.48	1.49	1.54	1.60
L2	1.00	1.30	1.44	1.58	1.54	1.53	1.58	1.66
L3	1.00	1.27	1.47	1.54	1.53	1.50	1.55	1.63
L4	1.00	1.24	1.46	1.48	1.44	1.45	1.52	1.57
L5	1.00	1.17	1.32	1.47	1.40	1.43	1.49	1.57
L6	1.00	1.16	1.37	1.43	1.40	1.34	1.51	1.50
L7	1.00	1.14	1.29	1.42	1.38	1.36	1.43	1.50
L8	1.00	1.13	1.31	1.43	1.37	1.37	1.44	1.52
L9	1.00	1.14	1.29	1.43	1.33	1.33	1.41	1.50

Table 7. TTS Normalized speedup

	1C	**2C**	**3C**	**4C**	**5C**	**6C**	**7C**	**8C**
L1	1.00	1.43	1.65	1.70	1.72	1.73	1.73	1.74
L2	1.00	1.34	1.57	1.61	1.64	1.64	1.65	1.68
L3	1.00	1.37	1.52	1.54	1.55	1.58	1.59	1.64
L4	1.00	1.33	1.47	1.53	1.53	1.53	1.51	1.54
L5	1.00	1.34	1.43	1.47	1.47	1.49	1.49	1.51
L6	1.00	1.30	1.36	1.42	1.44	1.43	1.44	1.46
L7	1.00	1.28	1.34	1.38	1.39	1.38	1.41	1.46
L8	1.00	1.26	1.41	1.43	1.44	1.44	1.45	1.46

L9	1.00	1.31	1.38	1.38	1.41	1.41	1.41	1.43

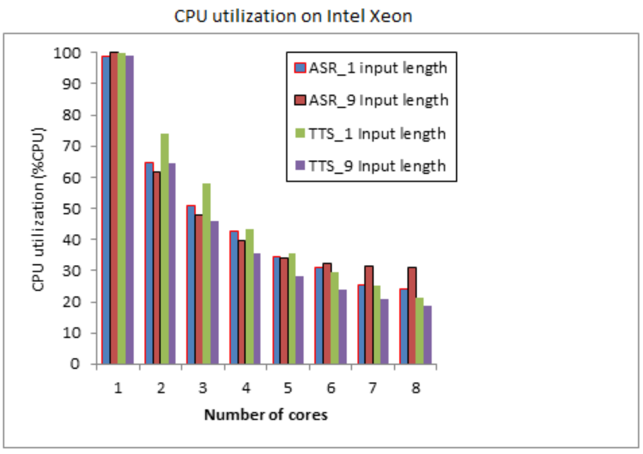


Figure 8. CPU utilization for ASR and TTS.

The experiment conducted on Intel core i-7 processor shows that when we enable 2 cores from different physical cores the execution time significantly decreases compare to enabling two hyperthreads on the same core. Keeping in mind that we have two physical cores and 2 hyper threads per each, this makes 4 logical cores on this processor. However, when we enable two cores that belong to same physical core, the execution time is similar as if only one core is enabled. Consequently, when three cores and four cores are enabled, we get the same execution time as if only 2 physical cores are enabled. In the worst case as observed in Figure 5 (i), when three and four cores are enabled, this adds overhead; hence introduce more delay to the response time. We observe that the execution time for ASR on 4 cores is higher that even on a single core. This is due to contention of shared resources when the parallel part tries to execute on separate logical cores in parallel. It is seen that hyperthreads share resources, for Intel core i-7, L1 cache and L2 cache are shared by hyperthreads that belong to the same physical core.

Furthermore, hyperthreads that belong to different physical cores share L3 cache. The speed up as shown by the slope value “b” in Table 4, decreases for almost a half from one core to four cores. The hyperthreading does not affect TTS because TTS executes sequentially, the execution time seems to be almost the same no matter how many cores or hyperthread are enabled. The value of “b” in Table 3, shows that the average speedup is about 26 %, i.e., for 3.37 seconds on one logical core to 2.80 seconds for four logical cores. This is Amdahl’s law, whereby the speed up of a program depends on how much parallel it is [20]. The part that increases threads is the garbage collector and this count almost 1.7 % to 2 percent of the whole work to be done. In this paper, neither ASR, though it has some parallel part, nor TTS benefits from hyperthreading. The main performance bottleneck is the access to the memory, whereby two virtual cores, i.e. hyperthreads contend to share L1 cache of 32 KB. Therefore this prevents taking advantage of enabling hyperthreading for these particular applications.

The results of ASR from Intel-xeon dual quadcore, shows that the number of threads that are generated, follows the number of enabled cores. This is shown in Table 5. The slope of the response time is shown in Table 1 and in Figure 5(ii). The slope (b value) shows the average speed up, it is about 1.54, from 1.28 to 0.83 for one and eight enabled cores respectively. Table 6 shows the normalized speed up, where we have the average of 1.56 for eight enabled cores. Both results from regression model and normalization tell us that the speedup is generally about 1.55, if we average both results. Consider the number of threads for ASR in Table 5 and the slope “b” in Table 1. At one core increase we see two threads increase, the garbage collector that counts for 2 to 3 percentage of the total work. The Scoring threads contribute to the speed up significantly. However, the speedup does not grow that fast as expected instead it slowly increases but gradually. There is about 7 % speed up at each core increase.

Though we cannot give strict conclusion based on CPU utilization only. It can help us to investigate and understand more the behavior of our programs. The CPU utilization in Figure 8 decreases from 100 % to 31% from 1 core to 8 cores shows that ASR consumes for example 31 % of 8 available cores. Additionally, the average speed up is from both regression and normalization is about 1.55 for 8 cores. Therefore, we can generalize our performance model to more cores; i.e., the right part of the matrix in Figure 4(ii). Increasing one core corresponds to 7 % speed up increase.

Moreover, based on the fitting line it is straightforward to predict the response time on the basis of exclusively multicore processors with hyperthreading disable or without it at all. This can be done for any input length for ASR when the number of cores varies from 1 up to 8. This corresponds to the lower left part of the matrix under the shaded part in Figure 4 (ii).

About TTS, the speed up as the slope “b” shows in Table 2 is averagely 1. 36, from 4.54 to 3.33 for one and eight enabled cores respectively. Table 7 shows the normalized relative speed up that stays constant as the number of cores increases. The relative normalized speed up is about 1.50. The speed up from regression and from normalization is averaged to 1.43. The CPU utilization that goes from 100% for one core to 17% for 8 cores, i.e., cores are not exploited as much as ASR does because ASR has a parallel part. However like ASR, as Figures 5(iii, iv) shows, the execution time for TTS increases linearly as the input length increases.

Comparing ASR and TTS, even though ASR has a parallel part, the speed up is slow. Obviously, as expected the speed up is better than for TTS. However, TTS also has a small speed up increase, i.e., OS, java virtual machine threads and the garbage collector contribute to that speed up. To validate the study, the precision measures R2 and MSE are used. In our case; the R2 is almost one and MSE is close to zero. This shows a strong correlation between the measured results and the predicted results. (See Tables 1 to 4). Furthermore, for the sake of time synchronization, during response time measurement, we used the time obtained from time command in Linux; as well as real wall clock time. In order to explore more the behavior of ASR and TTS on multicore processor, the CPU utilization (see Figure 8), the normalized relative speed up and the number of threads with respect to enabled cores are also measured.

The relation between results of measured and predicted is observed from analysis of variance regression output that shows a good fit between measured and predicted results. Moreover, the prism value computed at 95% level of confidence is far less than 0.05 which value rejects the standard null hypothesis that says that there is no correlation between studied variables.

To further validate our study; we used the most common performance metric for voice based application. It is the response time that is measured during the experiment. The same metric is used in predicting model. That model is the linear regression; it is common for predicting model. For the sake of variety, 2 types of multicore processors with different numbers of cores are used in the experiments.

To generalize the performance model, the response time as a function of input length, can be predicted for further length than the experiment range, this is shown by the straight line that indicates the linear increase of the response time as input length grows. Therefore, the linearity of the response time with respect to input length is general for any input length. This generalization is applied on multicore processor that does not apply hyperthreading. E.g., Intel-xeon dual quad core. However, the multicore processor with hyperthreading enabled is not generalizable. As we observed when running experiment on Intel core i-7. To extrapolate the response time for further number of cores, we base our analysis on the number of threads. Generally, the speed up of ASR increases of 7 % at each increase of a core, i.e., one thread for scoring and another for garbage collection. For TTS has found to be sequential, it means that there is no expected speed up for higher number of cores. However, there are some systems 'parts such as java virtual machine and the garbage collector that benefit from cores increase, therefore the total speed up increases.

3.2.6 Conclusions

This paper presents a performance evaluation of open source speech recognition and text to speech, i.e., Sphinx-4 and FreeTTS-1.2.2 when these are implemented on multicore processors. The hardware used are, Intel core i-7 quad cores, and Intel-xeon dual quadcore.

In order to use the multicore processor economically, ASR and TTS should be investigated and redesigned. Free and open source help in this. The experiment results on Intel core i-7 cannot be generalized because of the hyperthreading which does not add up to the ASR parallel part neither to TTS. In this paper, the generalization can only be done on multicore with no hyperthreading. The measured average response time is studied as a function of the input length and the number of cores. The number of threads, the relative speed up, and CPU utilization are also measured in order to investigate the performance in detail. The results from the performance model when varying the load length reveals that both ASR and TTS have a linearity property of the response time with respect to the input length; this property tells us that it is relatively easy to generalize the performance model to any input length.

The response time from one number of cores to another is described by the changes of the regression equation coefficient, i.e., the slope of the speedup increase. Both ASR and TTS apply garbage collector which is inherently parallel. It counts for almost 2 % of the total work. Moreover, ASR has acoustic scoring part that runs in parallel; the number of threads is always updated dynamically with the number of cores respectively. The speed up is about 7% per one core increase. And this could be used to extrapolate in order to predict for further than experiment range of cores. The speed up increase from the program itself is trivial because the latter is sequential. However, some system's part like OS, java virtual machine thread and garbage collector benefits from the core increase, hence the TTS total response time get a little increase of about 5% at each core increase. In this paper, the generalization is application specific, i.e., Sphinx-4 and FreeTTS -1.2.2. To open source development community, based on the findings, it is important to always update the source code in order to accommodate current technology.

References

[1] J. R. Lewis, P. M. Commarford, and C. Kotan, 'Web-Based Comparison of Two Styles of Auditory Presentation: All TTS versus Rapidly Mixed Tts and Recordings', in Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 2006, vol. 50, pp. 723–727.

[2] C. Niyizamwiyitira, L. Lundberg, and M. Svahnberg, 'Evaluation of Voice-driven Web Application Architecture', in 2012 Eighth International Conference on Signal Image Technology and Internet Based Systems (SITIS), 2012, pp. 555 –562.

[3] 'FreeTTS 1.2 - A speech synthesizer written entirely in the Java(TM) programming language'. [Online].
Available: http://freetts.sourceforge.net/docs/index.php. [Accessed: 10-May-2013].

[4] 'Sphinx-4 - A speech recognizer written entirely in the Java(TM) programming language'. [Online].
Available:http://cmusphinx.sourceforge.net/sphinx4/#source. [Accessed:08-Jul-2013].

[5] B. Schauer, 'Multicore processors–A necessity', ProQuest Discovery Guides1–14, 2008.

[6] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, 'Sphinx-4: a flexible open source framework for speech recognition', Sun Microsystems, Inc., Mountain View, CA, USA, 2004.

[7] H.-L. Lou, 'Implementing the Viterbi algorithm', Signal Processing Magazine, IEEE, vol. 12, no. 5, pp. 42–52, 1995.

[8] K. You, J. Chong, Y. Yi, E. Gonina, C. J. Hughes, Y.-K. Chen, W. Sung, and K. Keutzer, ‘Parallel scalability in speech recognition’, IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 124–135, 2009.

[9] S. Yoshizawa, N. Wada, N. Hayasaka, and Y. Miyanaga, ‘Scalable architecture for word HMM-based speech recognition and VLSI implementation in complete system’, IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 53, no. 1, pp. 70–77, 2006.

[10] X. Chen, C. Xu, R. P. Dick, and Z. M. Mao, ‘Performance and power modeling in a multi-programmed multi-core environment’, in Proceedings of the 47th Design Automation Conference, 2010, pp. 813–818.

[11] W. Walker, P. Lamere, and P. Kwok, ‘FreeTTS: a performance case study’, Sun Microsystems, Inc., Mountain View, CA, USA, 2002.

[12] V. S. Adve, Analyzing the behavior and performance of parallel programs, vol. 1201. Citeseer, 1993.

[13] A. S. Namin, M. Sridharan, and P. Tomar, ‘Predicting Multi-Core Performance: A Case Study Using Solaris Containers’, in Proceedings of the 3rd International Workshop on Multicore Software Engineering, ser. IWMSE, 2010, vol. 10, pp. 18–25.

[14] P. Lamere, P. Kwok, E. Gouvea, B. Raj, R. Singh, W. Walker, M. Warmuth, and P. Wolf, ‘The CMU SPHINX-4 speech recognition system’, in IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong, 2003, pp. 2–5.

[15] ‘Sphinx-4 Transcriber Demo’. [Online]. Available: http://cmusphinx.sourceforge.net/sphinx4/src/apps/edu/cmu/sphinx/demo/transcriber/README.html. [Accessed: 10-May-2013].

[16] ‘Alice’s Adventures in Wonderland (Project Gutenberg)’. [Online]. Available: http://www.cs.cmu.edu/~rgs/alice-table.html. [Accessed: 10-May-2013].

[17] ‘Home—Project Kenai’. [Online]. Available: http://visualvm.java.net/. [Accessed: 08-Jul-2013].

[18] J. O. Rawlings, S. G. Pantula, and D. A. Dickey, Applied regression analysis: a research tool. Springer, 1998.

[19] J. Miles and M. Shevlin, Applying regression and correlation: A guide for students and researchers. SAGE Publications Limited, 2001.

[20] M. D. Hill and M. R. Marty, ‘Amdahl’s law in the multicore era’, Computer, vol. 41, no. 7, pp. 33–38, 2008.

Chapter Four

Section 1

Published as C. Niyizamwiyitira and L. Lundberg, “Real-time scheduling of multiple virtual machines,” International journal of Computers and their applications (IJCA), Vol. 24, No. 3, pp.91-109, Sept. 2017

Real-time scheduling of multiple virtual machines

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 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 each VM in isolation. We also introduce an overhead model that makes it possible to find VM periods that minimize the processor utilization.

Keywords

Real-time Virtual Machine; Real-time scheduling; Hard deadlines; VM overhead; VM period

4.1.1 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], [2]. 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 [3] 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.

4.1.2 Background and Related Work

Real-time scheduling theory (or non-virtualized systems shows that the minimum processor utilization for which a 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 [4], and 53.2% for two processor systems and then down to as little as 37.5% for systems with infinitely many processors [5]. Consequently, compared to multiprocessor systems, the processor utilization is in general 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 on 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 focus on real-time hypervisor scheduling framework for Xen [6], [7]. There are a number of results concerning so called proportional-share schedulers [8]–[10]. These results looked at a real-time application that runs inside an operating system process. The proportional-share schedulers divides 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 [11], 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 [12]. 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 [13], 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 [14], [15] 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 schedule 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 [16], [17] 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 [16]. 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 uni-processor environment) show that one can guarantee the schedulability for more cases when the entire work-load is taken into consideration [18].
In [19] 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 [20] 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 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 [22] the authors propose a mechanism to schedule soft real-time systems, which provide 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, hard real-time system with strict deadline is considered.
In [23], [24] the authors present a model that accounts for the overhead, in compositional hierarchical scheduling for uniprocessor however, the entire

work-load was not considered. In [25] 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 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 [18]. In this paper we use the idea from the age constraint approach for scheduling real-time tasks in the VMs.
In [3] 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 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.

4.1.3 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 are 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 $T_{i,j}$ [26]. 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 $C_{i,j}$ [26].

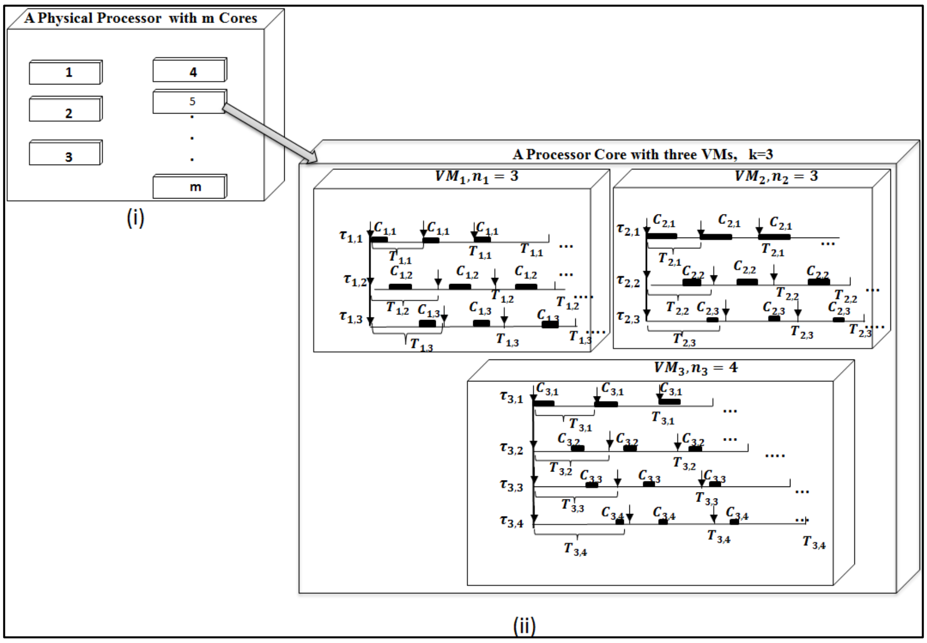

A Physical Processor with m Cores
1
2
3
4
5
m
(i)
A Processor Core with three VMs, k=3
$VM_1, n_1 = 3$
$VM_2, n_2 = 3$
$VM_3, n_3 = 4$
(ii)

Figure 1. (i) A Physical processor with *m* cores, and (ii) three Virtual Machines on a processor core.

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 *VMᵢ* executes at least C_{VM_i} time units every T_{VM_i} period.
In [3] 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 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 [3]. 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}.

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 VMi executes at least C_{VM_i} time units every T_{VM_i} period.
In [3] 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 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 [3]. 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}.

4.1.4 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.

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

In order to obtain $R_{i,j}$ from Equation 1, we need to use iterative numeric methods [26]. In Figure 2, we look at one VM in isolation [3]. The main result in [3] 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.

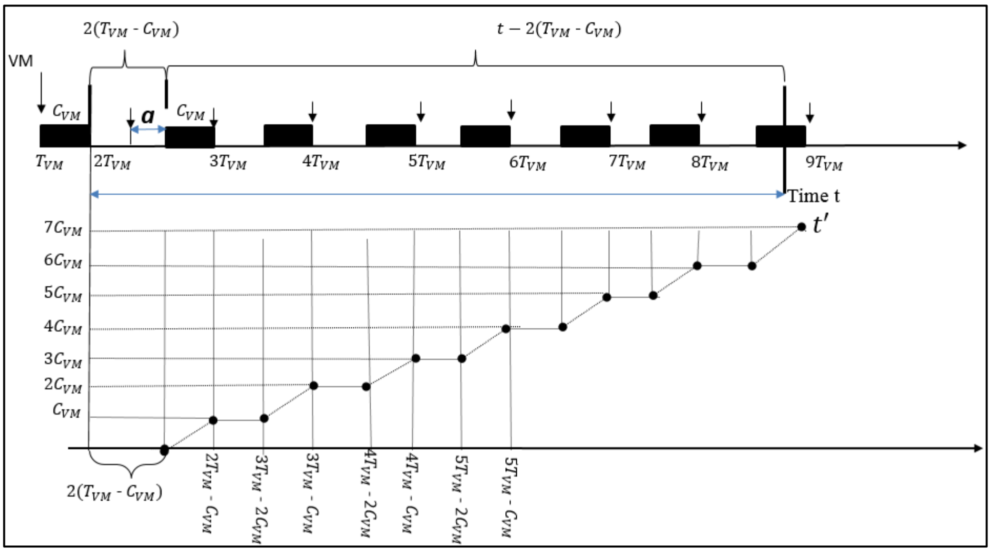


Figure 2. The worst-case scenario for an isolated VM

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).

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

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:

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

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 $\left((2+n)T_{VM} - 2C_{VM}), nC_{VM}\right)$ to$\left((2+n)T_{VM} - C_{VM}), (n+1)C_{VM}\right)$ for any *n* = 0, 1, 2…, (see Figure 2). As a result,

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

The horizontal line that connects two consecutive segments represents the end of 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 [27] 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}.

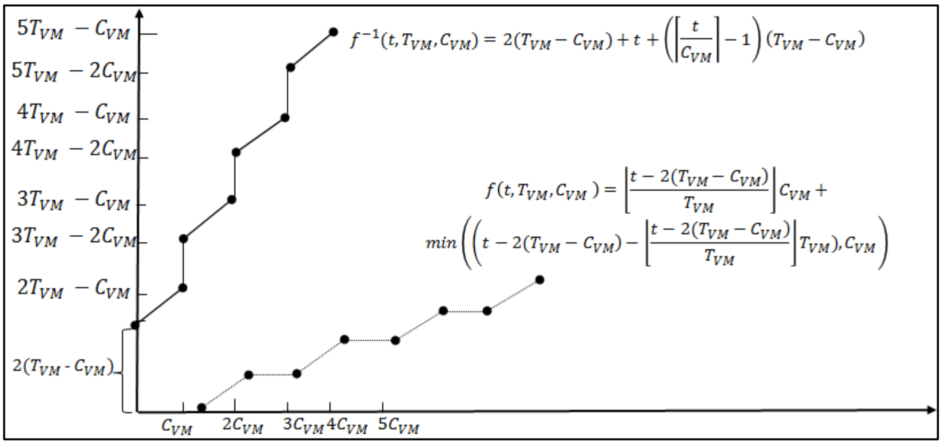


Figure 3. The (pseudo-)inverse function for an isolated VM (i.e., the pessimistic case).

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 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 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 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 $T_{VM_1} - C_{VM_1}$

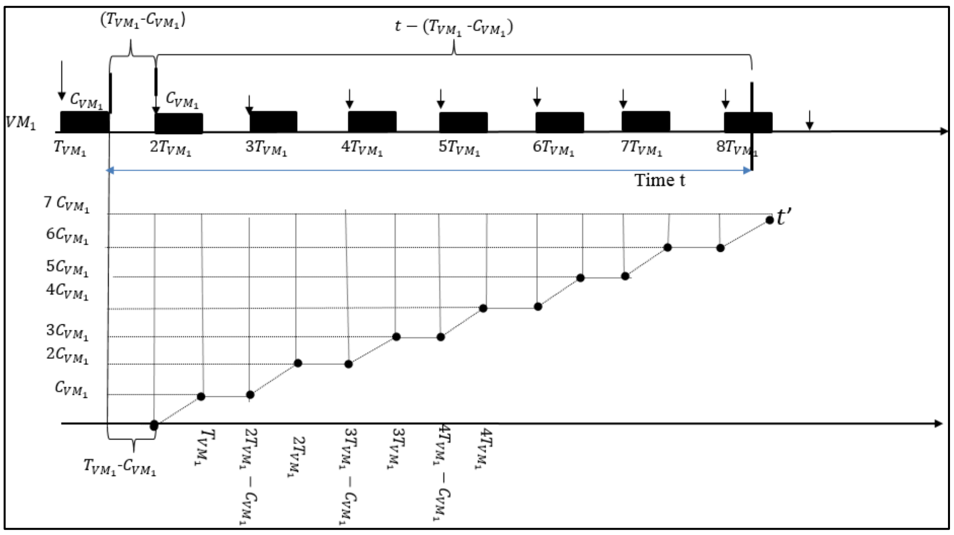


Figure 4. The worst-case scenario for VM_1.

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'* denotes 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,

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

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

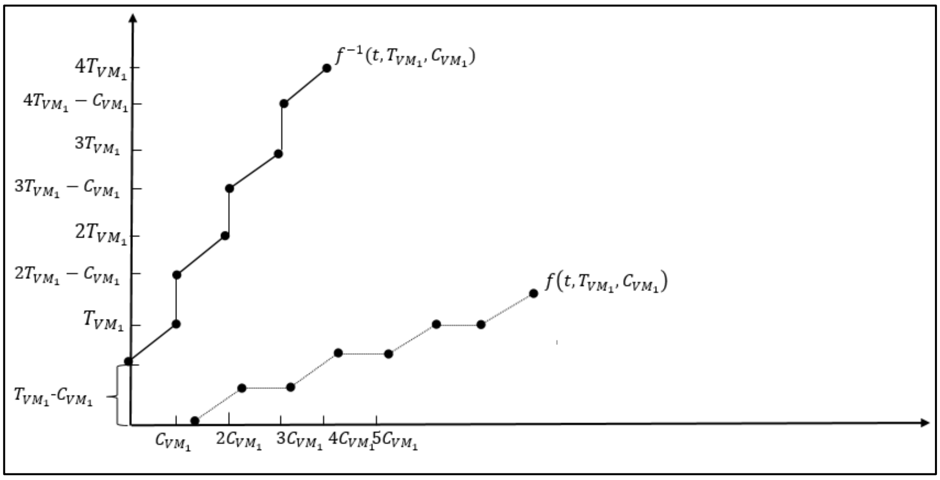


Figure 5. The (pseudo-)inverse function for VM_1.

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

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 $(T_{VM_1} - C_{VM_1})$ 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 $r_{1,j} = f^{-1}(R_{1,j}, T_{VM_1}, C_{VM_1})$

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

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

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.

$$R_{VM_2} = C_{VM_2} + \left\lceil \frac{R_{VM_2}}{T_{VM_1}} \right\rceil C_{VM_1} \tag{12}$$

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 $\left\lfloor \frac{t-(T_{VM_2}-2C_{VM_2}+R_{VM_2})}{T_{VM_2}} \right\rfloor$.

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

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

For fixed T_{VM_2}, C_{VM_2} and R_{VM_2}, Figure 6 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 $\left(\left((n+1)T_{VM_2} - 2C_{VM_2} + R_{VM_2}\right), nC_{VM_2}\right)$ to $\left(\left((n+\right.\right.$

$1)T_{VM_2} - C_{VM_2} + R_{VM_2}), (n+1)C_{VM_2}\Big)$ for $n = 0, 1, 2\ldots.$

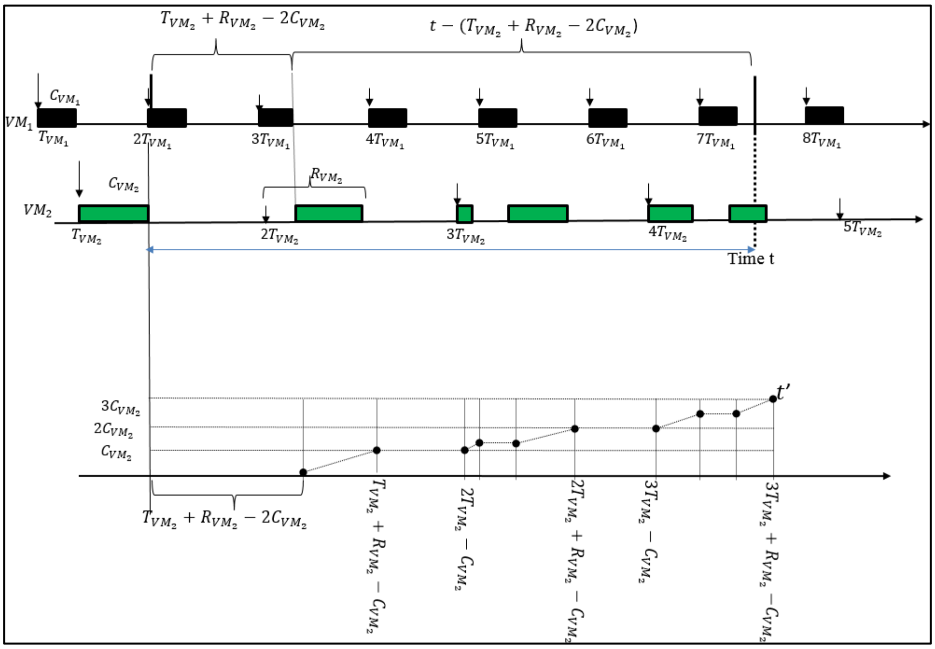


Figure 6. The worst-case scenario for VM_2.

As a result,

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

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

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

$$1\Big)\,(T_{VM_2} - C_{VM_2}) \tag{15}$$

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

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

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

$$r_{2,j} = f^{-1}\big(R_{2,j}, T_{VM_2}, C_{VM_2}, R_{VM_2}\big) \leq T_{2,j} \tag{17}$$

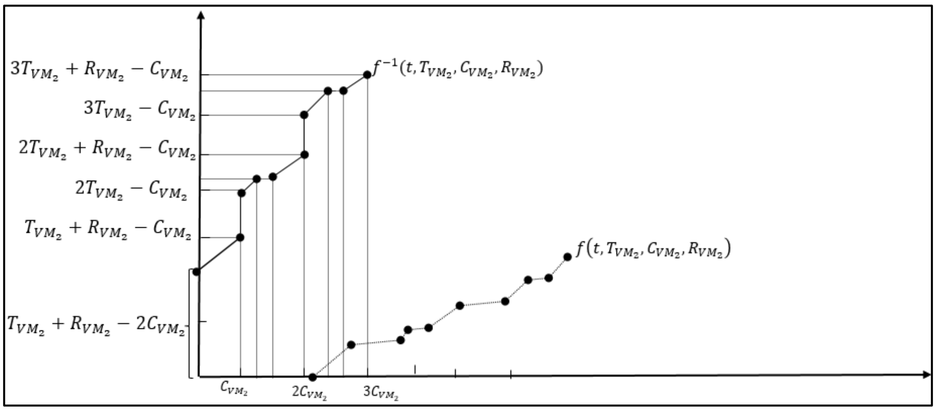


Figure 7. The (pseudo-)inverse function for VM_2

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.

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

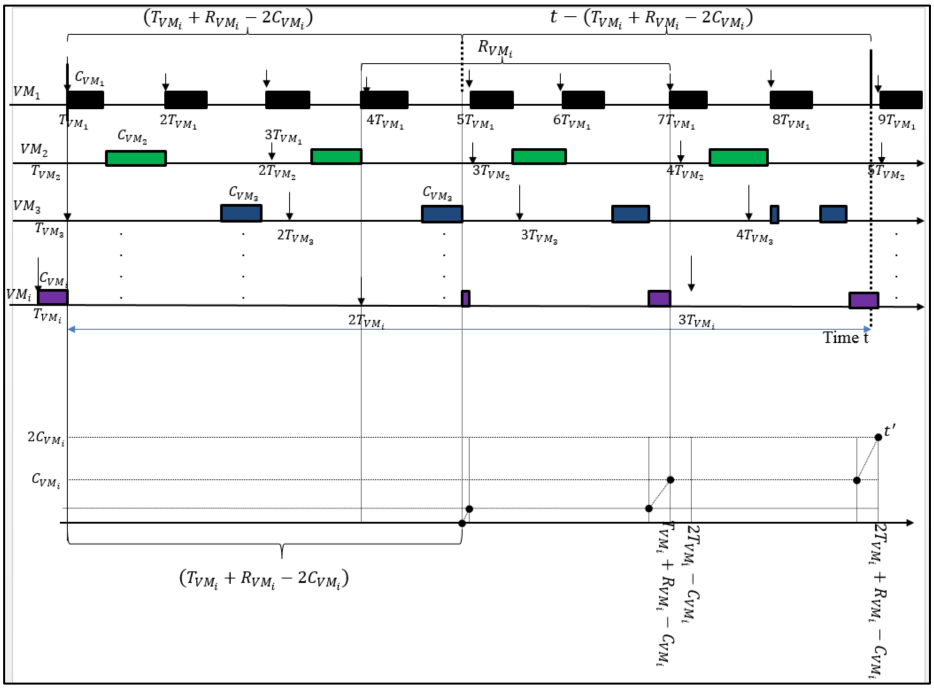


Figure 8. The worst-case scenario for VM_i.

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

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

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 $\left(\left((n+1)T_{VM_i} -\right.\right.$

$C_{VM_\mathrm{i}} + R_{VM_\mathrm{i}}), (n+1)C_{VM_\mathrm{i}}\big)$ for n = 0, 1, 2…., As a result,

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

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.

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

with $0 \le i \le k$ 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) $r_{i,j} = f^{-1}\left(R_{i,j}, T_{VM_i}, C_{VM_i}, R_{VM_i}\right)$.

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

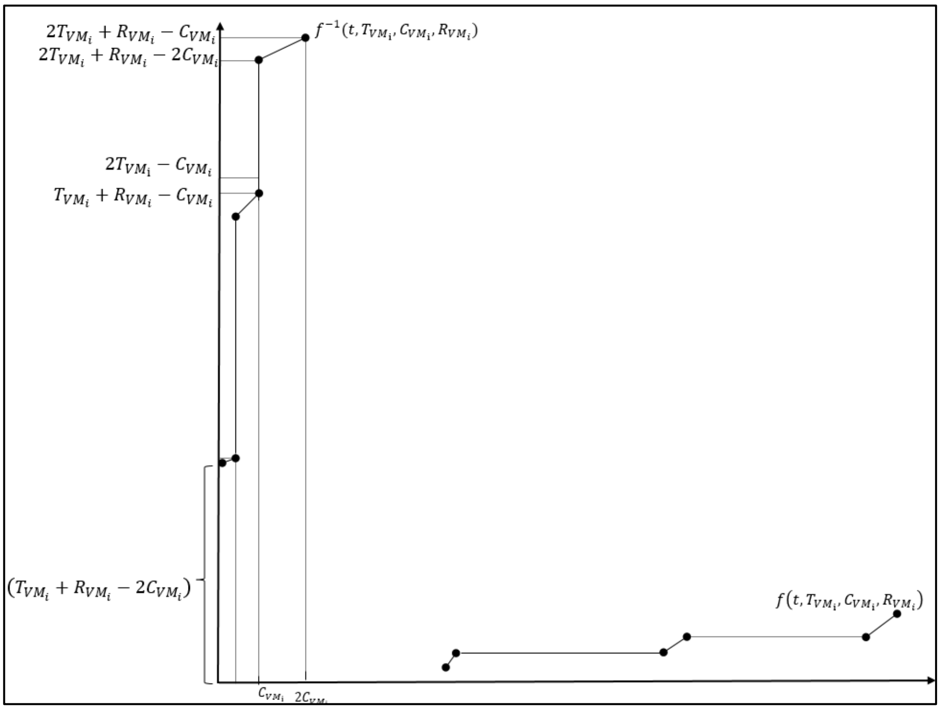

$2T_{VM_i} + R_{VM_i} - C_{VM_i}$
$2T_{VM_i} + R_{VM_i} - 2C_{VM_i}$
$f^{-1}(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$
$2T_{VM_i} - C_{VM_i}$
$T_{VM_i} + R_{VM_i} - C_{VM_i}$
$(T_{VM_i} + R_{VM_i} - 2C_{VM_i})$
$f(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$
C_{VM_i}
$2C_{VM_i}$

Figure 9. The (pseudo-) inverse function for VM_i.

4.1.5 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 [23], [24]. 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 Fig 10).

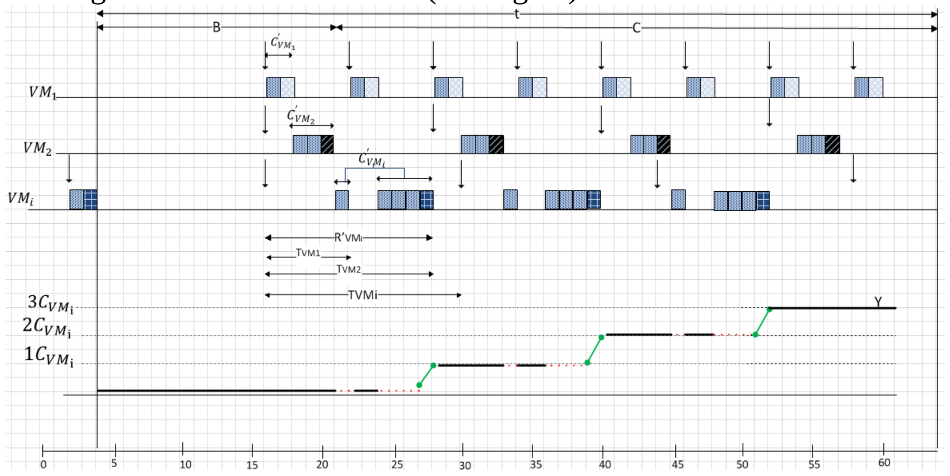
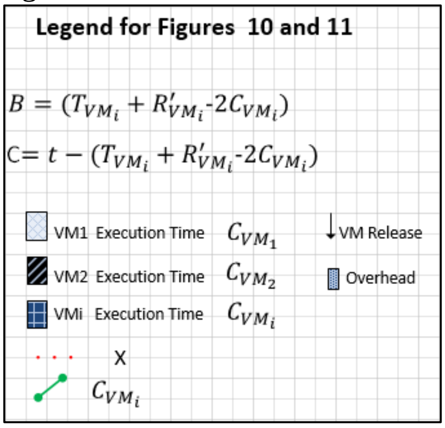


Figure 10. The worst-case scenario for VM_i with overhead.



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}		

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. Figure11 shows the pseudo-inverse function for VM_i with overhead. The inflated execution time is given by Equation 22

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

And the inflated response time is given by Equation 23

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

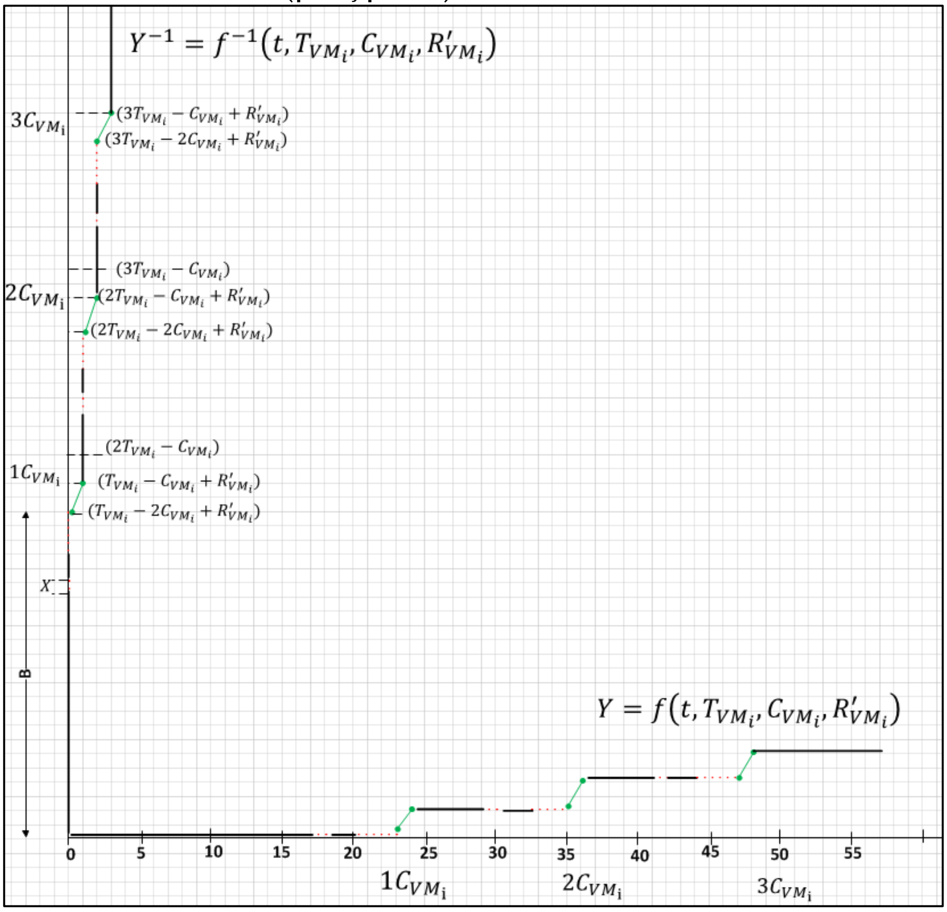


Figure 11. The (pseudo-)inverse function for VM$_i$ with overhead.

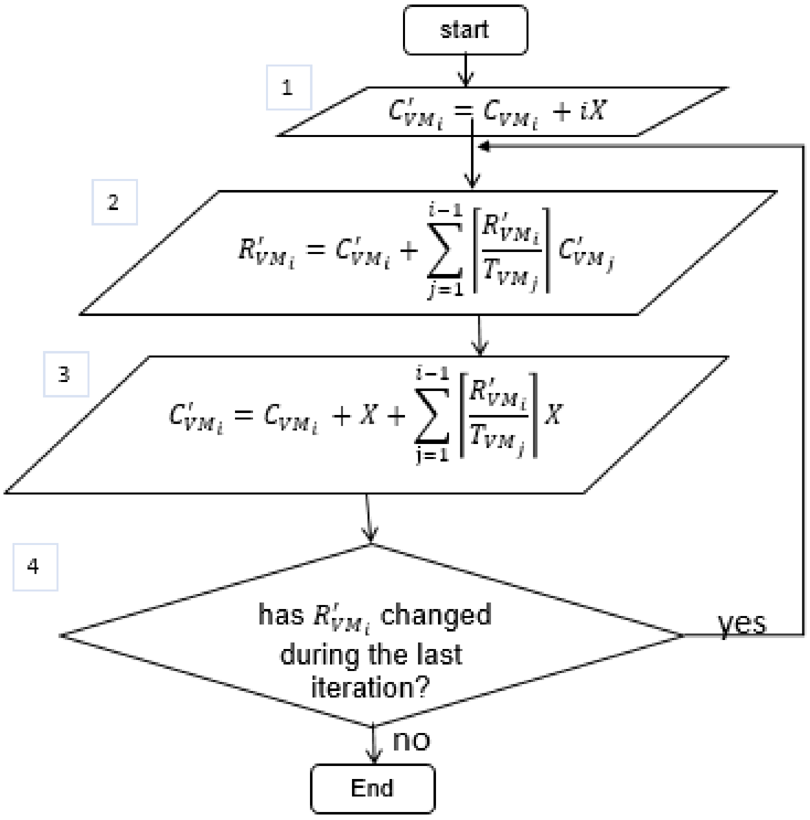

start
1
$C'_{VM_i} = C_{VM_i} + iX$
2
$R'_{VM_i} = C'_{VM_i} + \sum_{j=1}^{i-1} \left\lceil \frac{R'_{VM_i}}{T_{VM_j}} \right\rceil C'_{VM_j}$
3
$C'_{VM_i} = C_{VM_i} + X + \sum_{j=1}^{i-1} \left\lceil \frac{R'_{VM_i}}{T_{VM_j}} \right\rceil X$
4
has R'_{VM_i} changed during the last iteration?
yes
no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

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

Equation 22 and 23 are solved using numeric iterative method, Figure12 describes this algorithm, below is the description of Figure12.

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 Figure10, 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,

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

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

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

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

$$1\Big)\,(T_{VM_i} - C_{VM_i}) \qquad (24)$$

Let C'_{VM_j} denotes the inflated worst-case execution time for VM_j. In that case we make 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.

4.1.6 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.

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. $C_{VM_1}/T_{VM_1} = 0.4$ and $C_{VM_1}/T_{VM_1} = 0.3$.

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

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

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

$R_{1,1} = C_{1,1}$, 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 [3] 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

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

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.

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

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

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

$T_{1,1} = 16$ 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 $\tau_{1,1}$ from 14 to 16, and task $\tau_{1,1}$ will be released again at time 16 and execute from 16 to 18. Since $C_{1,2} = 1$ 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 $C_{1,3} = 4$. 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) $r_{2,1} = f^{-1}(R_{2,1}, T_{VM_2}, C_{VM_2}, R_{VM_2}) \leq T_{2,1} = 28$ (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 at $T_{VM_2} - 2C_{VM_2} + R_{VM_2} = 27$ 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), (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 *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.

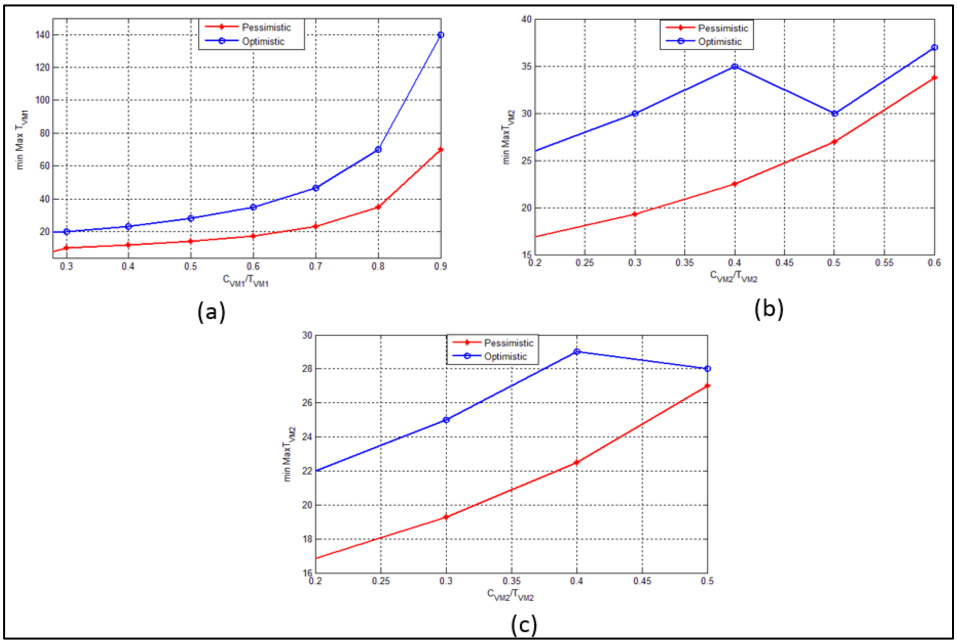


Figure 13. (a) C_{VM_1}/T_{VM_1} versus minimum of the max T_{VM_1}, (b) C_{VM_2}/T_{VM_2} versus minimum of the max T_{VM_2} when $C_{VM_1}/T_{VM_1} = 0.3$, (c) C_{VM_2}/T_{VM_2} versus minimum of the max T_{VM_2} when $C_{VM_1}/T_{VM_1} = 0.4$

If we consider the case where we have knowledge about one VM only, we have to make 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.

4.1.7 Simulations

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 $VM_1, VM_2,…, VM_8$ 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 Uunifast algorithm [28]. 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 higher 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 [29], [30].
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).

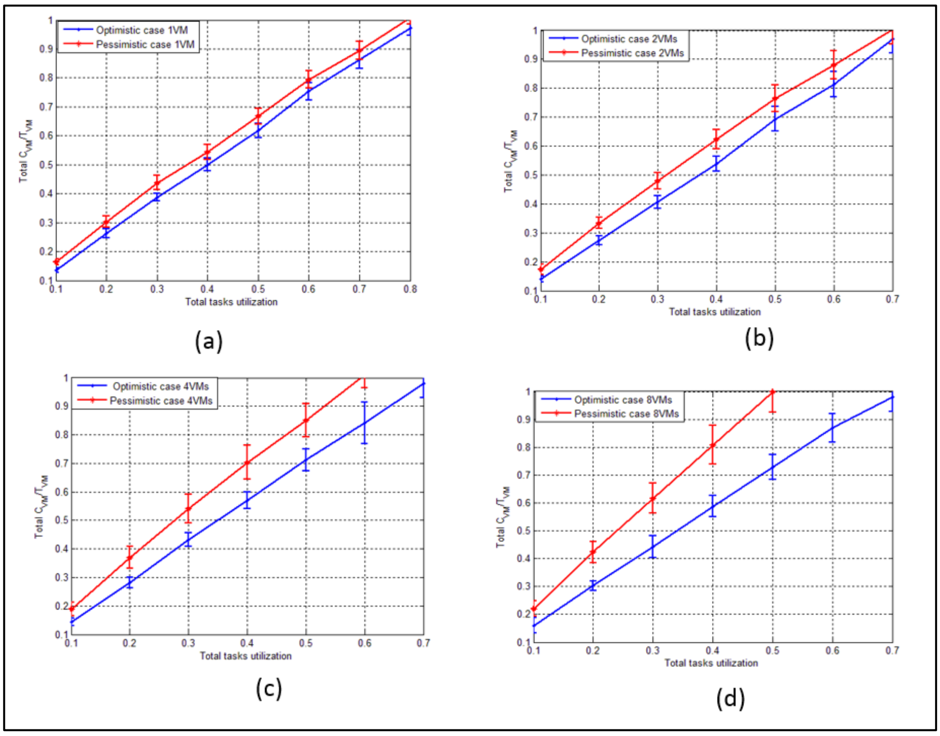


Figure 14. (a) C_{VM}/T_{VM} versus Total utilization for 1 VM, (b) C_{VM}/T_{VM} versus Total utilization for 2 VMs,

(c) C_{VM}/T_{VM} versus Total utilization for 4 VMs, (d) C_{VM}/T_{VM} versus Total utilization for 8 VMs

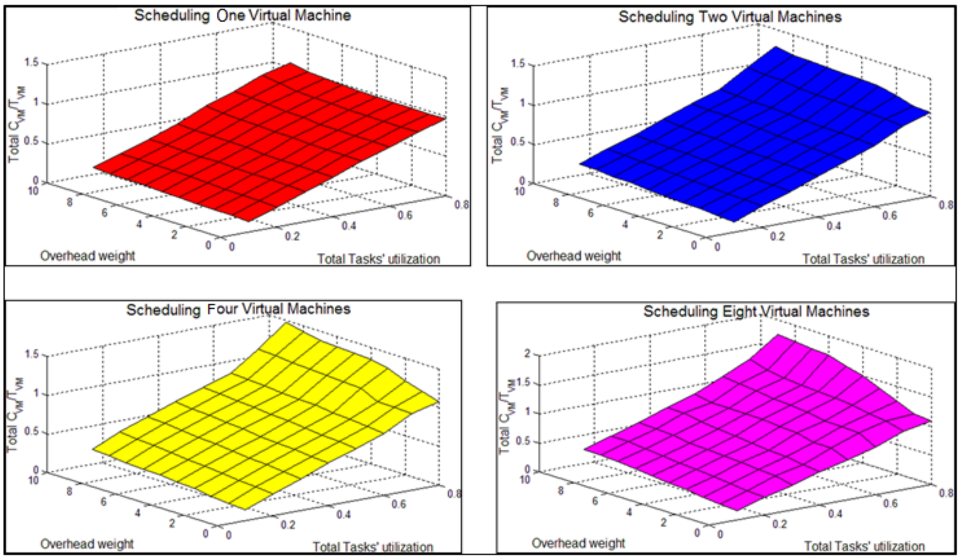


Figure 15. C_{VM}/T_{VM} versus Total utilization for optimistic case.

4.1.8 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 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 optimistic method saves resource than 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.

References

[1] 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, vol. 4, no. 2, pp. 189–193, 2012.

[2] L. Abeni and T. Cucinotta, “Efficient virtualisation of real-time activities,” in Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on, pp. 1–4, 2011.

[3] L. Lundberg and S. Shirinbab, “Real-time scheduling in cloud-based virtualized software systems,” in Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies, pp. 54–58, 2013.

[4] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” Journal of the ACM (JACM), vol. 20, no. 1, pp. 46–61, 1973.

[5] L. Lundberg, “Analyzing fixed-priority global multiprocessor scheduling,” in Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings, pp. 145–153, 2002.

[6] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, “Supporting soft real-time tasks in the xen hypervisor,” in Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, New York, NY, USA, pp. 97–108, 2010.

[7] S. Xi, J. Wilson, C. Lu, and C. Gill, “RT-Xen: Towards real-time hypervisor scheduling in Xen,” in 2011 Proceedings of the International Conference on Embedded Software (EMSOFT), pp. 39–48, 2011.

[8] K. J. Duda and D. R. Cheriton, “Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler,” in ACM SIGOPS Operating Systems Review, vol. 33, pp. 261–276, 1999.

[9] 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,” in Real-Time Systems Symposium, 1996., 17th IEEE, pp. 288–299, 1996.

[10] J. Nieh and M. S. Lam, “A SMART scheduler for multimedia applications,” ACM Transactions on Computer Systems (TOCS), vol. 21, no. 2, pp. 117–163, 2003.

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

[12] B. Lin and P. A. Dinda, “Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling,” in Proceedings of the 2005 ACM/IEEE conference on Supercomputing, pp. 8, 2005.

[13] X. Feng and A. K. Mok, “A model of hierarchical real-time virtual resources,” in Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE, pp. 26–35, 2002.

[14] I. Shin and I. Lee, “Periodic resource model for compositional real-time guarantees,” in Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE, pp. 2–13, 2003.

[15] I. Shin and I. Lee, “Compositional real-time scheduling framework with periodic model,” ACM Transactions on Embedded Computing Systems (TECS), vol. 7, no. 3, pp. 30, 2008.

[16] G. Lipari and E. Bini, “A methodology for designing hierarchical scheduling systems,” Journal of Embedded Computing, vol. 1, no. 2, pp. 257–269, 2005.

[17] R. I. Davis and A. Burns, “Hierarchical fixed priority pre-emptive scheduling,” in Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International, p. 10–pp, 2005.

[18] L. Lundberg, “Utilization based schedulability bounds for age constraint process sets in real-time systems,” Real-Time Systems, vol. 23, no. 3, pp. 273–295, 2002.

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

[20] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, “Realizing compositional scheduling through virtualization,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th, pp. 13–22, 2012.

[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] T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi, “Providing performance guarantees to virtual machines using real-time scheduling,” in Euro-Par 2010 Parallel Processing Workshops, pp. 657–664, 2011.

[23] S. Chen, L. T. Phan, J. Lee, I. Lee, and O. Sokolsky, “Removing abstraction overhead in the composition of hierarchical real-time systems,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE, pp. 81–90, 2011.

[24] L. T. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky, “Overhead-aware compositional analysis of real-time systems,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th, pp. 237–246, 2013.

[25] 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,” in Real-Time Systems Symposium (RTSS), 2013 IEEE 34th, pp. 1–10, 2013.

[26] 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.

[27] 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, vol. 58, no. 2, pp. 279–286, Feb. 2009.

[28] E. Bini and G. C. Buttazzo, “Measuring the performance of schedulability tests,” Real-Time Systems, vol. 30, no. 1–2, pp. 129–154, 2005.

[29] M. Kutil, P. Sucha, R. Capek, and Z. Hanzalek, “Optimization and scheduling toolbox,” Matlab—Modelling, Programming and Simulations, pp. 239–260, 2010.

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

Chapter Four

Section 2

Published as C. Niyizamwiyitira and L. Lundberg, “Period assignment in real-time scheduling of multiple virtual machines,” in Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems, 2015, pp. 180–187.

Period assignment in real-time scheduling of multiple virtual machines

Abstract

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. In order to save more resources, we propose to schedule VMs that contains hard real-time application in an optimistic manner by considering VMs as a whole instead of being isolated. Based on the properties of the real-time applications inside the VMs, we obtain (worst-case) execution times and periods for the VMs containing the real-time applications. A VM's period will be chosen according to the requirement of the application that runs inside that VM. Through simulation, we investigate the impact of the length of different VMs' periods whether the VM context switching overhead is present or not. The tradeoff between resources consumption and period's length is presented. The results show that longer periods are better at handling higher overhead even with higher number of VMs that share the same hardware.

Keywords

Virtualization; Real-time scheduling; Hard- deadlines; Virtual Machine scheduling; VM period assignment.

4.2.1 Introduction

As virtualization emerges, moving a real-time system with hard deadlines to a virtualized environment where a number of Virtual Machines (VMs) share the same physical computer becomes an important but also 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. A typical application is the Infrastructure as a Service (IaaS) in cloud computing where providers map virtual resources to physical resources in a scalable fashion, i.e., a VM is allocated resources dynamically according to the workload, work type, overhead, etc. This include real-time systems such as image and video processing, navigation systems, voice and object recognition, etc.

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. Each VM contains a hard real-time application. We assume that for each core, the identities of the VMs that share that core are known and every VM is aware of its collocated VMs. We also assume that these VMs are scheduled to the physical processor core using static priorities and the server scheme is a periodic server. In such a system there will be scheduling on two levels [1]. The first level is traditional real-time scheduling of the tasks within a VM; two popular optimal 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. The second level is scheduling of VMs by the hypervisor; the hypervisor is a VM monitor that controls several VMs on the same physical hardware. In [2] the study looked at each VM in isolation, where the hypervisor offers the resource to each individual VM assuming each VM is isolated from the other; this could lead to excessive use of CPU resources (this is referred as overprovisioning in Datacenter) and low network throughput due to inter-VM communication overhead [3],[4]. In this paper we consider that VMs will be aware of their co-resident VMs and run according to the assigned priorities.

On one hand, when we use longer periods for the VMs, we must increase the processor resources in order to make sure that all tasks will meet their deadlines [2]. On the other hand, short periods cause significant switching overhead between VMs. Therefore, we need to calculate a VM's period T_{VM} that minimizes the VM computation resource consumption (C_{VM}/T_{VM}, where C_{VM} is the time that VM must execute during each period T_{VM}) and at the same time reduce the number of context switches. The contribution of this paper is to investigate how T_{VM} affects the VM's need for computation resource (C_{VM}/T_{VM}), as a result, the tradeoff between C_{VM}/T_{VM} and T_{VM} is presented. We use heuristic method to find T_{VM} that makes the tasks inside a VM to be schedulable. Previous studies assumed that T_{VM} equals to a half of the shortest task's period [5]. The methodology to conduct this study is simulation. The contribution of this paper is the investigation of VMs' periods length in the presence of context switch and scheduling overhead, this is important since the period of a VM can be allocated in elastic manner. The measurements show that the overhead is compensated by higher VM's period (T_{VM}) especially for higher number of VMs that share the same computing resource. The motivation of this paper is scheduling real-time applications with hard deadline in cloud computing on level of IaaS, where VMs share the same hardware can run real-time applications with hard deadline. In this case VMs are allocated a period and execution time in such a way that the application meets deadline.

The rest of this paper is as follows: Section 2 gives the background and related work, Section 3 contains the problem definition, Section 4 explains the simulation study, Section 5 presents the results and discussion, and Section 6 draws conclusions.

4.2.2 Background and Related Work

Real-time scheduling theory (for non-virtualized systems) shows that the minimum processor utilization for which a periodic real-time system can miss a deadline, using fixed priority scheduling, decreases as the number of processors increases, e.g., 69.3% for one processor systems [6] (using RMS). And 53.2% for two processors system and then down to as little as 37.5% for systems with infinite number of processors [7]. Consequently, compared to multiprocessor systems, the processor utilization is in general 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 on modern multi-core hardware server. In order to provide high hardware utilization we expect that many VMs may share the same processor core. In [3], [4], it has been proven that aware co-residents VMs that share the memory consume lower resource than isolated VMs. In [2] 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 VMs are treated in isolation. In [5], authors improve this technique by proposing a method that treats all VMs as a whole and provides period (T_{VM}) and execution time (C_{VM}) to all VMs that share the same processor. That method assumes that the entire workload, consisting of multiple task sets (one task set in each VM, is known and T_{VM} equals to a half of the shortest task's period. In this paper, we investigate the performance and implications of using different periods' length when assigning periods to VMs.

Previous studies show that when we use longer periods for the VMs, we must increase the processor resource in order to make sure that all tasks will meet their deadlines. On the other hand, short periods cause significant switching overhead between VMs [2]. Studies have been done in adjusting tasks period in real-time systems; in [8]–[10] the authors proved that adjusting task's period improved the schedulability of taskset at the same time reducing the computation resources requirement. We also want to adjust VMs period in order to avoid using excessive CPU resources.

In [1], [11], the authors proposed a variant constant bandwidth server algorithm to schedule multiple VMs when RM or EDF schedulers were used for tasks inside the VM. Based on KVM they assign parameters to servers for given real-time applications so that the application is schedulable. They came up with a pair (Q, P) where Q is the maximum execution time and P is the server period that can make the application inside a server to be schedulable. Whereas this paper investigates how to assign periods to multiple VMs, given the periods and computational times of the tasks that run inside the VM while considering the interaction of all VMs that share the same processor core.
A periodic resource model was proposed for compositional scheduling theory by grouping multiple tasks' requirements into a single task requirement in a hierarchical order [12]. The model is related to our study since we also use hierarchical scheduling. By use of a periodic resource model, 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 [13]. 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 causes additional overhead [14]. And, according to Lundberg [7], when using fixed priority scheduling, the processor utilization decreases as the number of processors increases. Multi-core VMs thus result in very low processor utilization. In this paper, we assume that each VM is bound to a single processor core.

4.2.3 Problem Definition

We consider the case when *k* VMs share the same processor core (see Figure 1(i)).
We assume that for each core the identities of the VMs that share a processor core are known. We also assume that these VMs are scheduled to the physical

core using static priorities. 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 (see Figure 1(ii)). A task $\tau_{i,j}$ is defined by its worst-case execution time $C_{i,j}$ and period $T_{i,j}$ [15]. Since we assume that the priority follows rate monotonic scheduling (RMS), the tasks are ordered such that $T_{i,j} \leq T_{i,j+1}$. This means that inside VM_i, task $\tau_{i,1}$ has the highest priority, we assume that the deadline $D_{i,j}$ equals to the period $T_{i,j}$. We need to calculate a period T_{VM} and an execution time C_{VM} for each VM that share a physical core, such that existing real-time programs will be schedulable in the VMs when each VM executes at least C_{VM} time units every T_{VM} period.

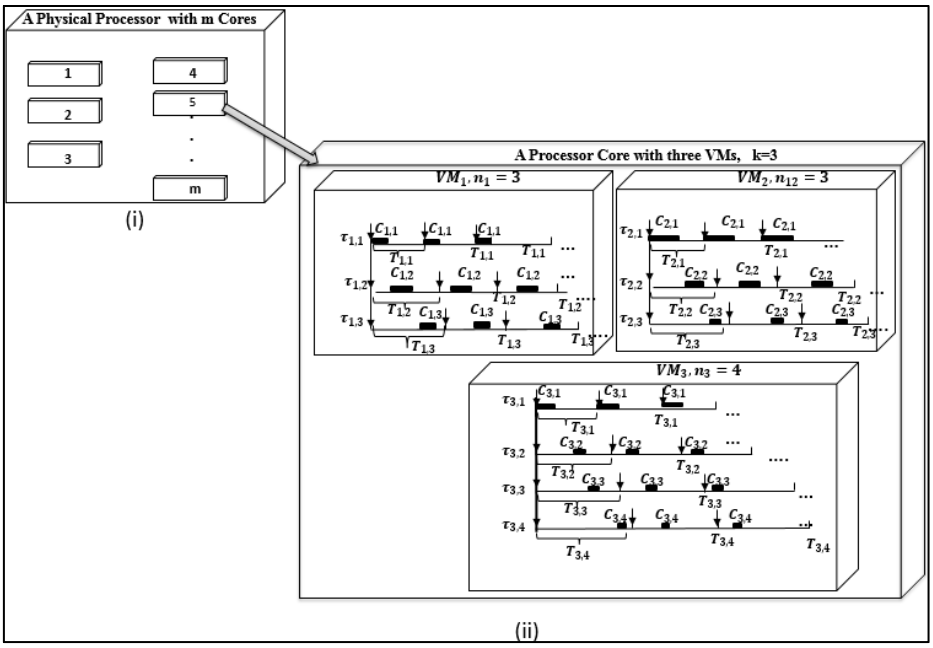

A Physical Processor with m Cores
1
2
3
4
5
m
(i)
A Processor Core with three VMs, k=3
$VM_1, n_1 = 3$
$VM_2, n_{12} = 3$
$VM_3, n_3 = 4$
(ii)

Figure 1. (i) A Physical processor with m cores, and (ii) three Virtual Machines on a core.

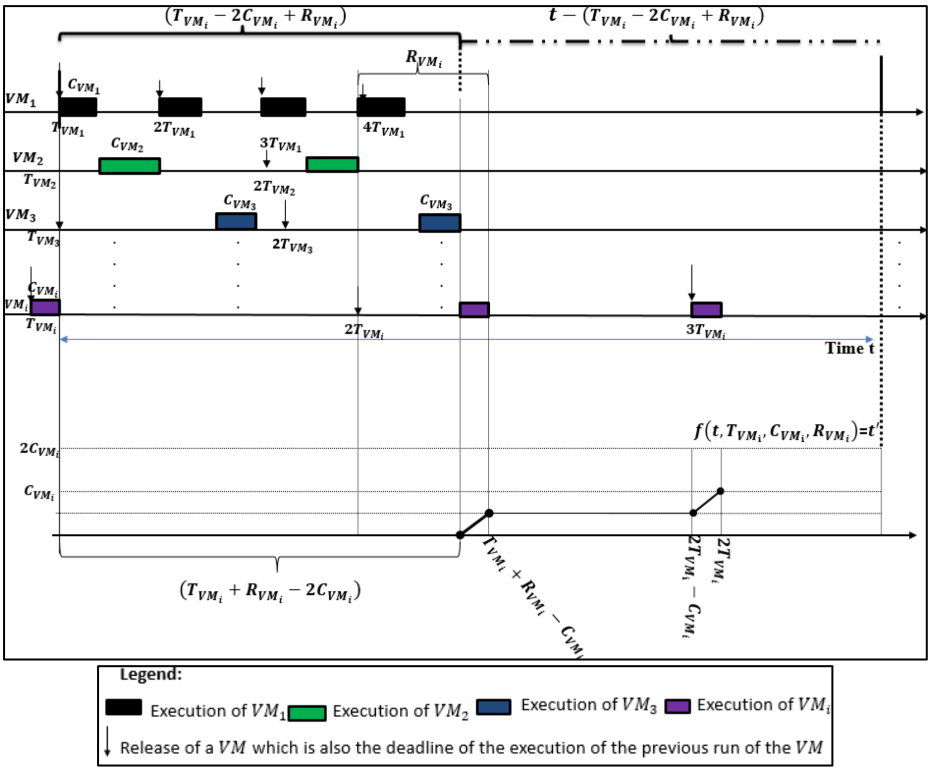


Figure 8. The worst-case scenario for *k* VMs.

In this paper we consider a factor (that helps to scale T_{VM}) that we multiply with the shortest tasks' period in order to get the T_{VM} for the VM. This factor helps to tune the VM's period, the so-called optimistic method looks at the entire set of VMs that share the same processor core. Figure 2 shows the runtime of *k* VMs, when they are scheduled using optimistic method. The VMs are scheduled to the physical core using static priorities that are ordered such that $T_{VM_i} \leq T_{VM_{i+1}}$. We consider a time period *t* which may extend over several periods T_{VM}. As Figure 2 shows, in the worst-case scenario *t* starts by a period
$(T_{VM_i} - 2C_{VM_i} + R_{VM_i})$ before the VM 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 1.

$$\left\lfloor \frac{t-(T_{VM_i}-2C_{VM_i}+R_{VM_i})}{T_{VM_i}} \right\rfloor \tag{1}$$

In Figure 2, let *t'* denote the minimum amount of time that the VM is running during a time period of length *t*. Then we have Equation 2

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

Equation 2 shows that *t'* is a function of four parameters $t' = f(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$.

For fixed T_{VM_i}, C_{VM_i}, and R_{VM_i} Figure 2 shows an increasing function $f(t, T_{VM_i}, C_{VM_i}, R_{VM_i})$ 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,…

In Figure 3 $f^{-1}(t, T_{VM}, C_{VM})$ is undefined during the time intervals when $f(t, T_{VM}, C_{VM})$ is flat. In [16] 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_i}, C_{VM_i}, and R_{VM_i}.

From Figure 3, the pseudo-inverse function which is the maximum time that VM is running during time *t* and that maps virtual time to (worst-case) real-time for VM_i is given in Equation 3

$$f^{-1}\left(t, T_{VM_i}, C_{VM_i}, R_{VM_i}\right) = \left(T_{VM_i} - 2C_{VM_i} + R_{VM_i}\right) + t + \left(\left\lceil \frac{t}{C_{VM_i}} \right\rceil - 1\right)\left(T_{VM_i} - C_{VM_i}\right) \tag{3}$$

With $0 \leq i \leq k$ and

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

From Equation 3 the worst-case response time $r_{i,j}$ for task $\tau_{i,j}$ is given by Equation 4.

$$r_{i,j} = f^{-1}\left(R_{i,j}, T_{VM_i}, C_{VM_i}, R_{VM_i}\right) \qquad (4)$$

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

The appropriate values of $r_{i,j}$ are found through numeric iteration methods. The condition for schedulability is that the worst-case response time must be less or equal to the task’s period ($r_{i,j} \leq T_{i,j}$).
The best case is when $R_{VM_i} = C_{VM_i}$, it means Equation 5.

$$f^{-1}\left(t, T_{VM_i}, C_{VM_i}\right) = \left(T_{VM_i} - C_{VM_i}\right) + t + \left(\left\lceil \frac{t}{C_{VM_i}} \right\rceil - 1\right)\left(T_{VM_i} - C_{VM_i}\right)$$
(5)
and it is for the first VM, because it has the highest priority.
The function in Equation 3 becomes equal to the pessimistic case when we treat all VMs in isolation where $R_{VM_i} = T_{VM_i}$ [2]. The worst-case response time $r_{i,j}$ (for task $\tau_{i,j}$),
$r_{i,j} = f^{-1}\left(R_{i,j}, T_{VM_i}, C_{VM_i}, R_{VM_i}\right)$ is derived from Equation 6.

$$f^{-1}\left(t, T_{VM_i}, C_{VM_i}, R_{VM_i}\right) = 2\left(T_{VM_i} - C_{VM_i}\right) + t + \left(\left\lceil \frac{t}{C_{VM_i}} \right\rceil - 1\right)\left(T_{VM_i} - C_{VM_i}\right) \qquad (6)$$

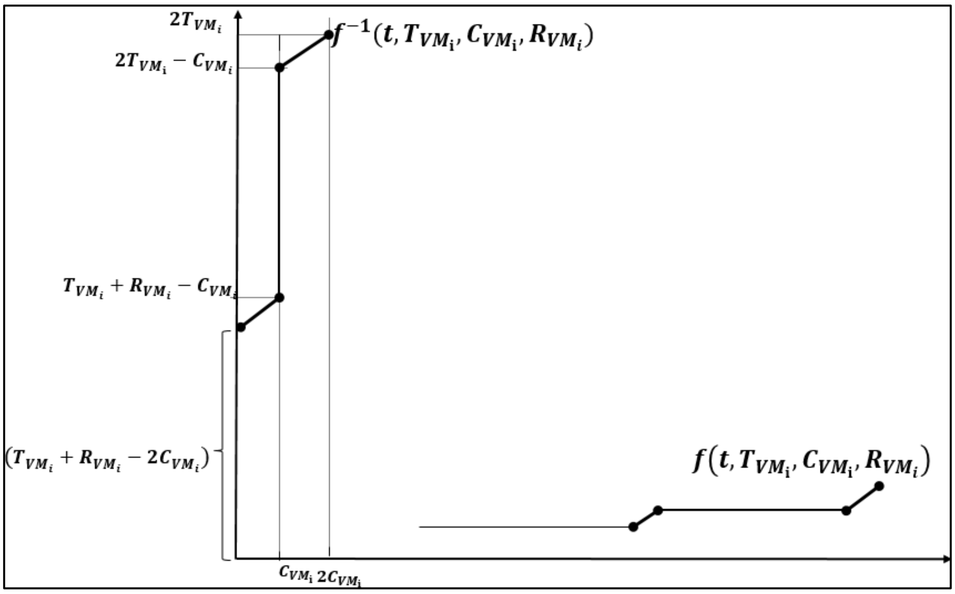


Figure 9. The pseudo-inverse function for *k* VMs.

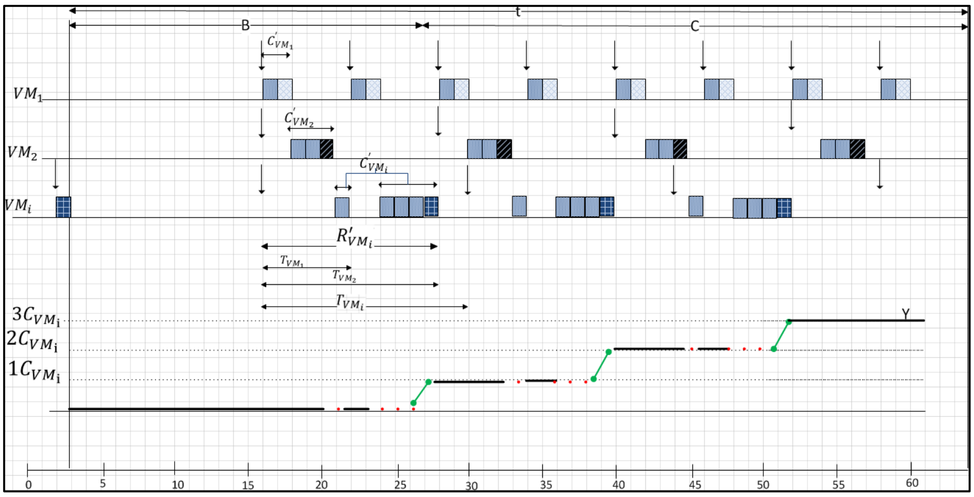


Figure 4. The worst-case scenario for VM_i with overhead.

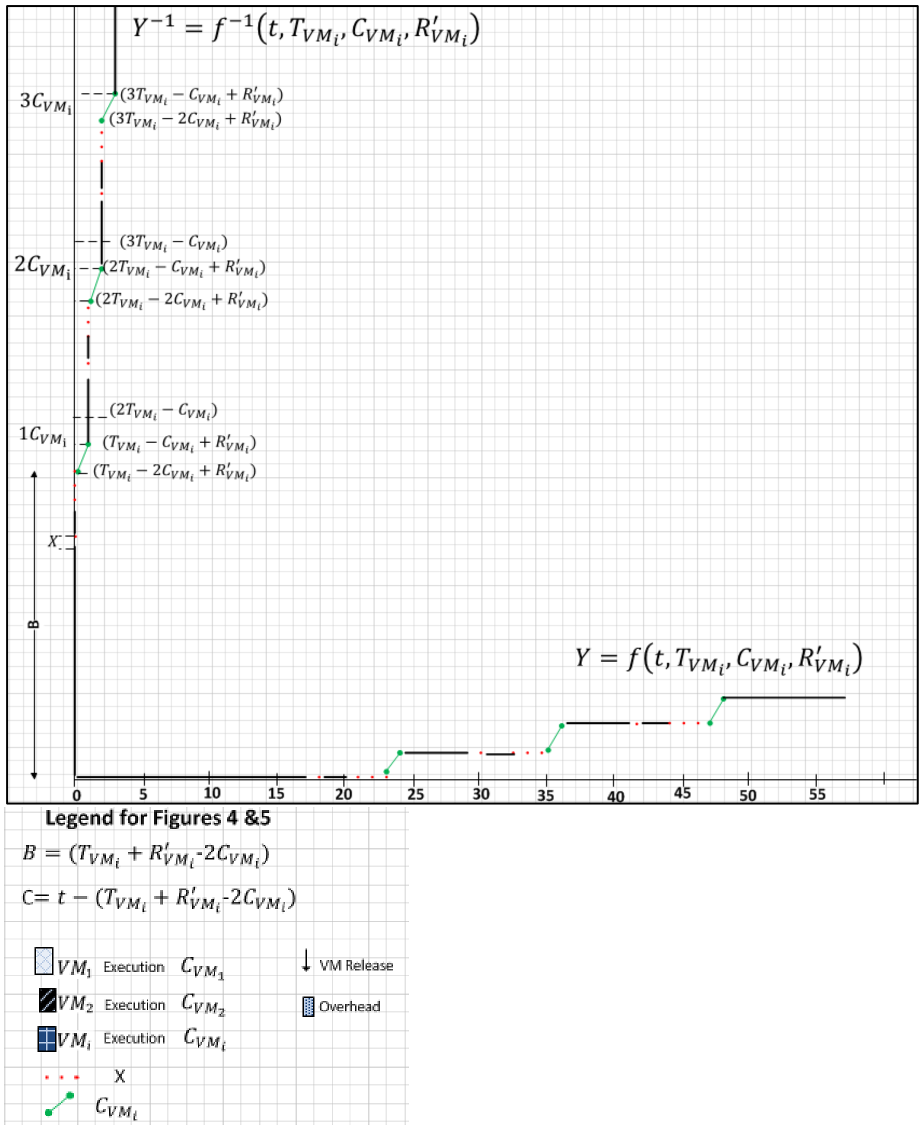


Figure 5. The pseudo-inverse function for VM_i with overhead.

4.2.4 Accounting for Overhead

In this paper we consider preemptive system. Whenever preemption takes place, different sources of overhead must be taken into account. Previous studies have looked at overhead in compositional real-time systems [17], [18]. 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 workload (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; 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 task execution times).

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 4, VM_i with $j < i$, has 4 X overhead, 1 X is the initial startup overhead of the VM, 2 X preemptions from VM_1 and 1 X from VM_2 in the worst-case scenario. The pseudo-inverse function to calculate task's the worst-case response time is given by Equation 7 where the inflated worst-case response time R'_{VM_i} for a VM is given in Equation 8 and the inflated worst-case execution time C'_{VM_i} is given in Equation 9. The corresponding pseudo-inverse function is given in the Figure 5. The condition for schedulability is that the worst-case response time must be less or equal to the task's period ($r_{i,j} \leq T_{i,j}$) with $r_{i,j} = f^{-1}\left(t, T_{VM_i}, C_{VM_i}, R'_{VM_i}\right)$.

$$f^{-1}\left(t, T_{VM_i}, C_{VM_i}, R'_{VM_i}\right) = (T_{VM_i} - 2C_{VM_i} + R'_{VM_i}) + t + \left(\left\lceil \frac{t}{C_{VM_i}} \right\rceil - 1\right)(T_{VM_i} - C_{VM_i}) \qquad (7)$$

$$R'_{VM_i} = C'_{VM_i} + \sum_{j=1}^{i-1} \left(\left\lceil \frac{R'_{VM_i}}{T_{VM_j}} \right\rceil C'_{VM_j} \right) \qquad (8)$$

$$C'_{VM_i} = C_{VM_i} + X + \sum_{k=1}^{i-1}\left(\left\lceil\frac{R'_{VM_i}}{T_{VM_k}}\right\rceil X\right) \qquad (9)$$

Equation 8 and 9 are solved using numeric iterative method as described in Figure 6.

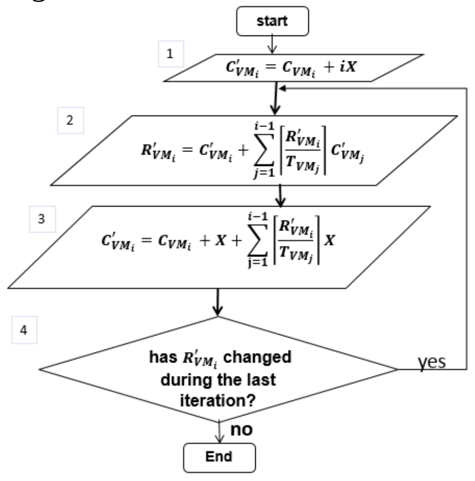

start
1
$C'_{VM_i} = C_{VM_i} + iX$
2
$R'_{VM_i} = C'_{VM_i} + \sum_{j=1}^{i-1}\left\lceil\frac{R'_{VM_i}}{T_{VM_j}}\right\rceil C'_{VM_j}$
3
$C'_{VM_i} = C_{VM_i} + X + \sum_{j=1}^{i-1}\left\lceil\frac{R'_{VM_i}}{T_{VM_j}}\right\rceil X$
4
has R'_{VM_i} changed during the last iteration?
yes
no
End

Figure 6. The algorithm flow chart to find C'_{VM_i} and R'_{VM_i}.

5. Inflated execution time is initialized, i.e., $C'_{VM_i} = C_{VM_i} + iX$
6. Inflated response time R'_{VM_i} is calculated using initial inflated execution time.
7. 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.
8. If R'_{VM_i} value does not change anymore, then we get the value for R'_{VM_i}, and for C'_{VM_i}.

Example in Figure 4, 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,

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

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

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

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

Therefore $R'_{VM_3} = 12$, $C'_{VM_3} = 5$
X (overhead weight) is arbitrary set to 1 since we are using synthesis tasks, but in the real world application, the overhead will take different weight. We make 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.

4.2.5 Simulation

The scheduling of tasks inside each VM uses RMS. We consider 8 programs that run in a 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,…, VM_8 respectively. These intervals show typical applications that have different deadline. We see that tasks'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 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 the tasks in each VM have a total utilization of 0.3. When we have total utilization of a program, we generate and distribute this total utilization to the 10 tasks using the Uunifast algorithm [19]. Each task's execution $C_{i,j}$ is then obtained by multiplying the utilization of the task with the task's period generated previously. We have used heuristic method by setting the VM's period ($\mathrm{T_{VM}}$) to be equal to the shortest tasks' period in the VM multiplied by a factor in the interval [0.1,..,1.25]. This interval is reasonable since the time elapsed before a VM can start its first execution in the first period must be less than the shortest tasks' period as long as the priority is based on rate monotonic (RM). Then we calculate the VM's hardware utilization, C_{VM}/T_{VM}. We repeated each unique case for different values of CPU utilization (for instance, one unique case is 4 VMs and a total utilization of 0.4) 20 times to be able to calculate average values and standard deviations. We used the Matlab scheduling toolbox TORSCHE (Time Optimization of Resources, SCHEduling) to schedule the first level, i.e., tasks inside VM [20], [21]. We repeated the simulation for different overhead weight (X values) X= [0,1,2,…,9] zero means the absence of overhead.

4.2.6 Results and Discussion

Figure 6 shows $\mathrm{C_{VM}/T_{VM}}$ for 1 VM for the total utilization of 0.2 and 0.6 of tasks inside the VM with respect to different period length. We see that for shorter period, in the absence of the overhead the resource utilization ($\mathrm{C_{VM}/T_{VM}}$) decreases. However, for higher overhead for example when the overhead equals to 9, $\mathrm{C_{VM}/T_{VM}}$ decreases for higher periods and it increases for shorter periods.

We observe that for 2 VMs, 4 VMs and 8 VMs (see Figures 7, 8 and 9) for higher overhead and shorter period, $\mathrm{C_{VM}/T_{VM}}$ increases to such extend that VMs may even become unschedulable. Due to paper size limitations, we only show the results for some scenarios in Figures 6 until 9 for the utilization 0.2 and 0.6 of tasks inside VMs. For the other utilizations, we also observe that $\mathrm{C_{VM}/T_{VM}}$ changes in the same manner. The results presented here are based on simulations, and may thus not give the same $\mathrm{C_{VM}/T_{VM}}$ value for all task sets of different applications. However, we expect that the results' trends will be the same, i.e., VMs will benefit from longer period especially for higher number of VMs, and for higher overhead as shown in Figures 6, 7, 8, and 9. E.g., suppose IaaS provider has to serve 3 customers who run hard real-time applications, the first customer needs to run a financial system that need to produce results in short time, the second needs to run a navigation system, and the third need to stream video. The provider needs to arrange these customers according to their deadlines, consequently the VMs will be allocated their period respectively. We know that 3 VMs for 3 customers will be allocated a processor in a timeshare manner. If navigation system has the shortest period then it will have the highest priority and it is allocated to VM1, if financial system has the second shortest period it is allocated to VM2, and video application will be allocated to VM_3. By using Equation 7, the VMs scheduler can dynamically determine how long VM period has to be in order to meet the application' deadline.

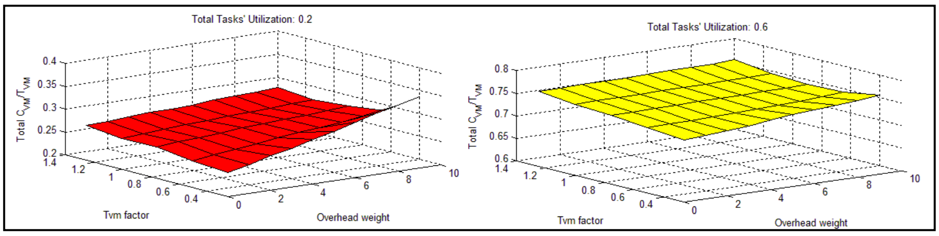


Figure 7. C_{VM}/T_{VM} for 1VM, for total tasks' utilization 0.2, and 0.6.

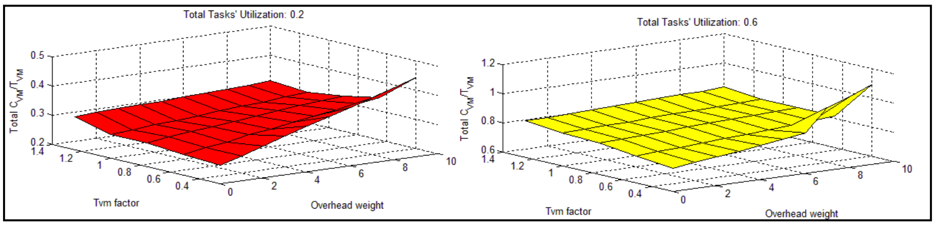


Figure 8. C_{VM}/T_{VM} for 2VMs, for total tasks' utilization 0.2 and 0.6.

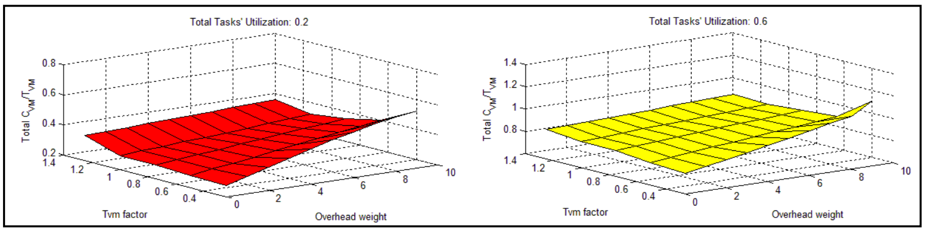


Figure 9. C_{VM}/T_{VM} for 4VMs, for total tasks' utilization 0.2 and 0.6.

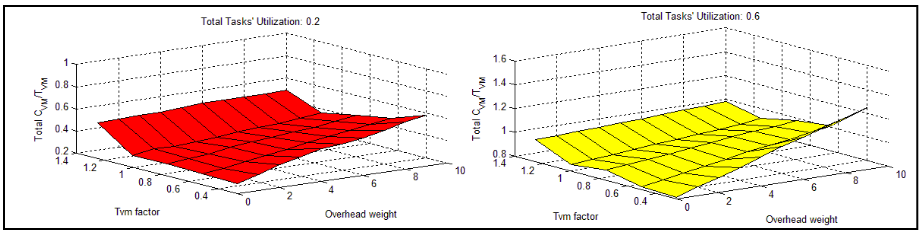


Figure 10. C_{VM}/T_{VM} for 8VMs, for total tasks' utilization 0.2 and 0.6.

4.2.7 Conclusions

In this paper we heuristically find different VMs' periods and investigate the impact of period length with respect to number of VMs and the overhead. We also investigated how the choice of factor that we use when we determine the T_{VM} based on the shortest tasks' period in the VM affects the processor resource. The best choice of T_{VM} will depend on the costs for VM switches and the periods of the tasks running inside the VMs. Our result quantifies the trade-off between CPU resource (C_{VM}/T_{VM}) and VM's period (T_{VM}) when selecting an appropriate T_{VM} where higher number of VM will benefits from using longer period especially in the presence of the overhead. Due to context switch, as the number of VMs increases, as we need longer period. The method described in this paper can be applied to allocate resources for real-time application using IaaS where virtual machine is a key element.

References

[1] L. Abeni and T. Cucinotta, "Efficient virtualisation of real-time activities," in Service-Oriented Computing and Applications (SOCA),

2011 IEEE International Conference, 2011, pp. 1–4.

[2] L. Lundberg and S. Shirinbab, “Real-time scheduling in cloud-based virtualized software systems,” in Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies, 2013, pp. 54–58.

[3] Y. Ren, L. Liu, Q. Zhang, Q. Wu, J. Wu, J. Kong, J. Guan, and H. Dai, “Residency-Aware Virtual Machine Communication Optimization: Design Choices and Techniques,” in Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, Washington, DC, USA, 2013, pp. 823–830.

[4] M. Kurtadikar, A. Patil, P. Toshniwal, and J. Abraham, “An Inter-VM Communication Model Supporting Live Migration,” in 2013 International Conference on Cloud Ubiquitous Computing Emerging Technologies (CUBE), 2013, pp. 63–68.

[5] C. Niyizamwiyitira and L. Lundberg, “Real-Time Systems Scheduling of Multiple Virtual Machines.” [Online]. Available: http://www.bth.se/people/cnw.nsf/pages/real-time-scheduling-virtualization. [Accessed: 09-Jun-2015].

[6] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” J. ACM, vol. 20, no. 1, pp. 46–61, 1973.

[7] L. Lundberg, “Analyzing fixed-priority global multiprocessor scheduling,” in Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE, 2002, pp. 145–153.

[8] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, “On task schedulability in real-time control systems,” in Real-Time Systems Symposium, 1996., 17th IEEE, 1996, pp. 13–21.

[9] D. Seto, J. P. Lehoczky, and L. Sha, “Task period selection and schedulability in real-time systems,” in Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE, 1998, pp. 188–198.

[10] T. Chantem, X. Wang, M. D. Lemmon, and X. S. Hu, “Period and deadline selection for schedulability in real-time systems,” in Real-Time Systems, 2008. ECRTS’08. Euromicro Conference on, 2008, pp. 168–177.

[11] T. Cucinotta, G. Anastasi, and L. Abeni, “Respecting Temporal Constraints in Virtualised Services.,” in COMPSAC (2), 2009, pp. 73–78.

[12] I. Shin and I. Lee, “Periodic resource model for compositional real-time guarantees,” in Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE, 2003, pp. 2–13.

[13] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, “Realizing compositional scheduling through virtualization,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th, 2012, pp. 13–22.

[14] S. Xi, C. Lu, C. Gill, M. Xu, L. T. Phan, I. Lee, and O. Sokolsky, “Global Real-Time Multi-Core Virtual Machine Scheduling in Xen,” Washington University Technical Report, Tech. Rep, 2013.

[15] A. Burns and A. J. Wellings, Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX. Pearson Education, 2001.

[16] E. Bini, T. H. C. Nguyen, P. Richard, and S. K. Baruah, “A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines,” IEEE Trans. Comput., vol. 58, no. 2, pp. 279–286, Feb. 2009.

[17] S. Chen, L. T. Phan, J. Lee, I. Lee, and O. Sokolsky, “Removing abstraction overhead in the composition of hierarchical real-time systems,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE, 2011, pp. 81–90.

[18] L. T. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky, “Overhead-aware compositional analysis of real-time systems,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th, 2013, pp. 237–246.

[19] E. Bini and G. C. Buttazzo, “Measuring the performance of schedulability tests,” Real-Time Syst., vol. 30, no. 1–2, pp. 129–154, 2005.

[20] M. Kutil, P. Sucha, R. Capek, and Z. Hanzalek, “Optimization and scheduling toolbox,” Matlab—Modelling Program. Simul., pp. 239–260, 2010.

[21] “TORSCHE Scheduling Toolbox for Matlab.” [Online]. Available: http://rtime.felk.cvut.cz/scheduling-toolbox/. [Accessed: 07-May-2014].

Chapter Five

C. Niyizamwiyitira and L. Lundberg, “A Utilization-based Schedulability Test of Real-time Systems Running on a Multiprocessor Virtual Machine, *Accepted to be published in the computer Journal, 2018*

A Utilization-based Schedulability Test of Real-time Systems Running on a Multiprocessor Virtual Machine

Abstract

We consider a real-time application that executes in a VM with multiple virtual cores. Tasks are scheduled globally using fixed-priority scheduling. In order to avoid Dhall's effect, we classify tasks into two priority classes: heavy and light. Heavy tasks have higher priority than light tasks. For light tasks we use rate monotonic priority assignment. We propose a utilization-based schedulability test. If the task set is schedulable, we provide an assignment of priorities to tasks. The input to the test is the task set, the number of cores in the VM, the period, deadline and blocking time for the VM. We evaluate how jitter, when scheduling VMs on the hypervisor level, affects the schedulability of the real-time tasks running in the VM. The schedulability of the real-time tasks in the VM decreases when the hypervisor jitter increases, but on the other hand the schedulability on the hypervisor level increases if we allow more jitter, i.e., there is a trade-off. Our results make it possible to evaluate this trade-off and take informed decisions when selecting scheduling parameters on the hypervisor level. Experimental evaluations show that the priority assignment used by our algorithm schedules more task sets than using rate monotonic (RM) priority assignment.

Keywords

Hard real-time scheduling; Multiprocessor utilization based schedulability test; Virtual multiprocessor scheduling; Global fixed priority scheduling; VM deadline

5.1 Introduction

Virtualization is a good way to increase resource utilization; it increases flexibility and scalability, and creates significant cost savings because of simplified system management. This is typically done by using so called virtual machines (VMs). In order to benefit from the cost-effectiveness of virtualization, real-time systems with hard deadlines are important applications to move to VMs. However, predicting if a real-time system is schedulable or not, is a challenging task. Virtualization makes it possible to run multiple VMs on one physical server, and it is possible to configure the VMs on multicore processor systems so that each VM uses all or a subset of the physical cores; many VMs may time share the same set of physical cores.

To guarantee schedulability of a real-time application inside a VM, we need to guarantee that the VM is executed a certain amount of time during a specified time period, i.e., we must provide a period and a minimum execution time when the VM runs the real-time tasks. This has been studied for VMs with one processor [1]. Guaranteeing real-time properties becomes more complex if the tasks are executed on a multiprocessor [2], and running the task set in a multiprocessor VM adds complexity compared to the normal (unvirtualized) multiprocessor case. In this paper we have developed a utilization-based schedulability test for a task set running in a VM with *m* virtual processor cores. In order to avoid Dhall's effect (to be explained later) we classify tasks into two priority classes, light tasks and heavy tasks (the state-of-the-art-technique for avoiding Dhall's effect).

A real-time application consists of a task set with deadlines. The CPU utilization U, is the fraction of processor time spent executing the task set $\Gamma = \{\tau_1, \dots, \tau_n\}$. U is calculated by summing up the contribution of each task τ_i ($u_i = C_i/T_i$ [3]; C_i is the worst case computation time for task τ_i and T_i is the period of τ_i). The overall system utilization factor for a uniprocessor system of a task set composed of *n* tasks is given in Equation 1

$$U = \sum_i^n u_i = \sum_i^n \frac{\mathrm{C_i}}{\mathrm{T_i}} \tag{1}$$

In the case of multiprocessor systems with *m* processors, the system utilization *US* is given in Equation 2

$$US = \frac{\sum_i^n u_i}{m} = \frac{\sum_i^n \frac{\mathrm{C_i}}{\mathrm{T_i}}}{m} \tag{2}$$

In this paper we define an algorithm that provides a utilization-based test that shows if a task set is schedulable or not. If the task set is schedulable the algorithm provides the priority for each task. The input to the algorithm is a task set Γ, the number of processor cores in the VM (m), the period for the VM (T_{VM}), the VM's deadline (D_{VM}), and the blocking time (Bp) during each period when the virtual machine does not have access to the underlying hardware (and thus also the computation time C_{VM} of the VM since $Bp = T_{VM} - C_{VM}$). See Figure 1 for an overview of the D_{VM}, T_{VM}, C_{VM}, and Bp parameters. These parameters and the problem definition, will be further explained in Section 3.

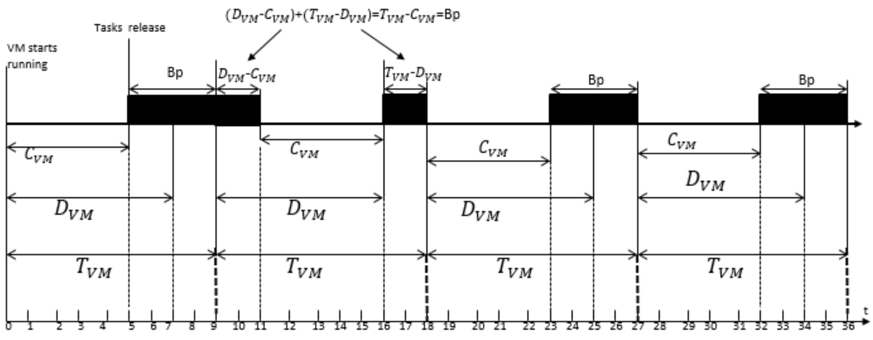


Figure 1. Scheduling a VM with deadline D_{VM} (T_{VM} = 9, D_{VM} = 7, Bp = 4, thus making C_{VM} = 5). The black parts represent the time intervals when the VM is blocked and does not have access to the hardware; the D_{VM} parameter controls the acceptable variation of the length of these periods (i.e., D_{VM} controls the acceptable jitter on the hypervisor level). If D_{VM} = C_{VM} (the minimum value of D_{VM}), all periods when the *VM* does not have access to the hardware must have the same length (*Bp*). If D_{VM} = T_{VM} (the maximum value of D_{VM}), the length of the periods when the *VM* does not have access to the hardware are allowed to vary between 0 and 2*Bp*. In general, the length of the periods when the *VM* does not have access to the hardware are allowed to vary between T_{VM} - D_{VM} and 2Bp - T_{VM} + D_{VM} = Bp - C_{VM} + D_{VM}.

This paper extends the study of utilization-based schedulability test of real-time systems on multiprocessor VM, where the authors proposed a scheduling algorithm that considered Bp, T_{VM}, m and the task set [4]. In the current paper, we consider an additional parameter; the deadline D_{VM} of the virtual machine during which the VM must have finished its execution time C_{VM}. The D_{VM}

parameter controls the acceptable amount of jitter on the hypervisor level; we will see that considering D_{VM} makes it possible to evaluate important trade-offs and take informed decisions when selecting scheduling parameters on the hypervisor level, also considering the effect of context switching overhead at the hypervisor level. Moreover, compared to the previous conference paper, the proofs are more complete, the main algorithm has been improved, and the paper contains experimental evaluations which compare the performance of our approach to traditional rate monotonic (RM) priority assignment. Additionally, the literature review has been extended considerably.

The rest of this paper is structured in the following way. Section 2 contains background and related work. Section 3 provides the problem definition. Section 4 describes the proposed algorithm. Section 5 discusses the VM deadline and the position of the blocking period. Section 6 presents the simulation, and finally, Section 7 presents our conclusions. Compared to the conference paper [4], Sections 5 and 6 are completely new, and large parts of the other sections have been rewritten and extended.

The rest of this paper is structured in the following way. Section 2 contains background and related work. Section 3 provides the problem definition. Section 4 describes the proposed algorithm. Section 5 discusses the VM deadline and the position of the blocking period. Section 6 presents the simulation, and finally, Section 7 presents our conclusions. Compared to the conference paper [4], sections 5 and 6 are completely new, and large parts of the other sections have been rewritten.

5.2 Background and Related Work

Liu and Layland have shown that for uni-processors rate monotonic (RM) priority assignment is optimal since no other fixed-priority scheme can schedule a task set that cannot be scheduled by RM [5]. They derived an upper bound of system utilization for guaranteed uni-processor scheduling for a task set of *n* tasks. Liu and Layland have shown that for uni-processors rate monotonic (RM) priority assignment is optimal since no other fixed-priority scheme can schedule a task set that cannot be scheduled by RM [5]. They derived an upper bound of system utilization for guaranteed uni-processor scheduling for a task set of *n* tasks.

$$U \leq n\left(2^{1/n} - 1\right) \tag{3}$$

The bound in Equation 3 tends to about 69% for infinitely large task set. This bound is, however, limited to uniprocessor systems. There are two basic approaches to extend preemptive task models to multiprocessor systems: partitioned and global scheduling. In the partitioned approach, a task is statically allocated to a particular processor core, i.e., a task is not allowed to migrate to another processor.

In the global approach, any tasks can execute on any processor. At every moment the m highest priority tasks are selected for execution on the m processors using preemption and migration if necessary. This is the most common way to schedule non real-time tasks on multiprocessor systems, and it is in many cases practical to handle non real-time and real-time tasks in a uniform manner.

The theory behind uniprocessor scheduling can be used for partitioned multiprocessor scheduling, e.g., the well-known uniprocessor schedulability test [6]. However, when it comes to global scheduling, the so called Dhall's effect shows that a direct application of the RM priority assignment scheme may result in failure to meet deadlines even for systems with very low multiprocessor utilization [7].

In Figure 2, eight tasks run on a multiprocessor with seven processors, Dhall's effect occurs if we assign priorities based on task periods without considering task utilization**,** for instance τ_8 misses its deadline even though the system utilization $US = \left((7 * 2/10) + 9/11\right)/7 \approx 0.317$. If we reduce C_i $(i = 1, \ldots, 7)$ to 1 and increase C_8 to 10, τ_8 will still miss the deadline, even though $US = \left((7 * 1/10) + 10/11\right)/7 \approx 0.230$.

Dhall's effect does not occur for liquid task sets [8] where each task has a very small utilization. If we consider any arbitrary task set, Dhall's effect can be avoided by dividing the tasks into two priority classes based on task utilization, namely heavy and light. The heavy tasks, i.e., the tasks with high utilization, get higher priority than the light tasks. In [9], the authors came up with a utilization bound of 33% for an infinitely large number of processors when using the heavy/light classification. Later on, an optimal bound of 37.5 % was derived by Lundberg in [2].

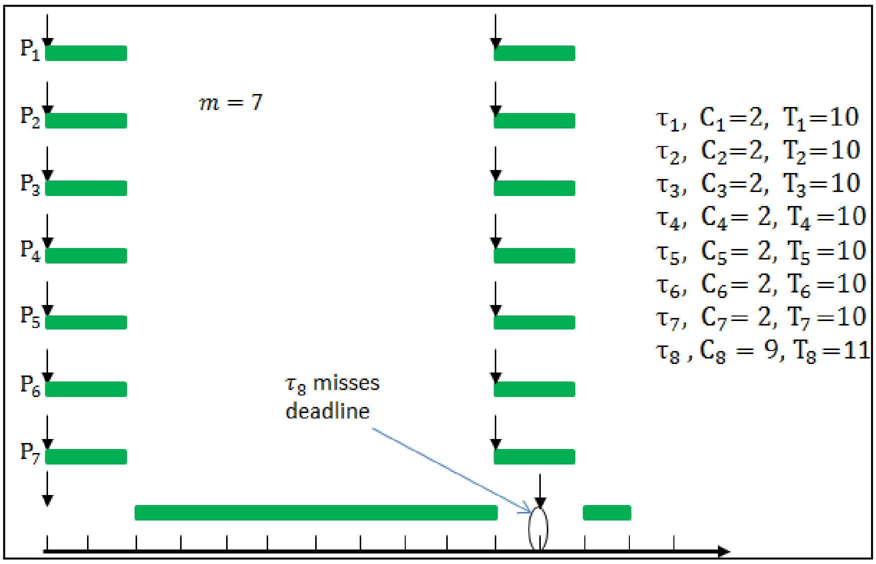

P1
P2
P3
P4
P5
P6
P7
$m = 7$
τ_8 misses deadline
τ_1, C_1=2, T_1=10
τ_2, C_2=2, T_2=10
τ_3, C_3=2, T_3=10
τ_4, C_4= 2, T_4=10
τ_5, C_5= 2, T_5=10
τ_6, C_6= 2, T_6=10
τ_7, C_7= 2, T_7=10
τ_8 , C_8 = 9, T_8=11

Figure 2. Illustration of Dhall's effect, adopted from [2]

In general, there are two kinds of schedulability tests: tests based on the utilization of the task set and tests based on more detailed analysis of the behavior of each task. The advantage of utilization-based tests is that they are fast and uncomplicated since they approximate the demand of the task set with the worst-case scenario for a given utilization; the drawback of utilization-based tests is that they are often pessimistic (since they consider the worst-case scenario) compared to the more detailed tests. In virtualized systems, one does not only need to consider the demand of the real-time tasks, one also needs to consider the supply of the processor resources during different time intervals. In this paper we approximate the demand of the task set with the worst-case scenario for a certain utilization, and compare this with the available supply of the processor resources. This approach gives the same advantages and drawbacks for the virtualized case as for the traditional unvirtualized case, i.e., compared to detailed models of both demanded and supplied times, our utilization based test is faster and less complicated, but also more pessimistic in some cases. The novelty of our proposed schedulability test is that it is utilization-based, while most of the previous work in the area is based on comparing demanded time and supplied time. We will show that, the computational complexity of our admission test algorithm is $O(mn)$, where n is the number of tasks and m is the number of virtual cores (processors) in the VM running the real-time tasks (in most real applications $m << n$). In the literature, there exist some studies that consider real-time virtual multiprocessor scheduling, below we present the ones that are most relevant for our study.

In [28] the authors present a schedulability test (admission test) based of the Parallel Supply Function (PSF). This test is valid for Earliest Deadline First (EDF) scheduling, which is a scheduling algorithm that uses dynamic task priorities; we consider static task priorities. The authors do not state the computational complexity of their admission test. However, the complexity of their test is clearly higher than the complexity of our test. PSF provides a detailed model of the supply function. The benefit with this is that the schedulability test does not become overly pessimistic. However, the price for the detailed model is that the analysis becomes complex.

Compared to PSF, the Multiprocessor Periodic Resource (MPR) [29] and the Generalized Multiprocessor Periodic Resource (GMPR) [27] models provide simpler supply functions. The price for the simplicity is increased pessimism in the schedulability analysis; the advantage is reduced complexity compared to PSF. However, both MPR and GMPR compare demanded time and supplied time, which makes them more complex than our utilization-based approach which simplifies the model for demanded time. The trade-off is again speed and simplicity (an advantage of the utilization-based approach) versus increased pessimism (a drawback of the he utilization-based approach).

An early model for comparing the demanded and supplied time in time-shared uni-processor system was the Bounded Delay Model (BDM) [30]. Shin and Lee have extended their compositional real-time scheduling framework (CSF) so that they can do schedulability analysis (or feasibility analysis as they call it) of BDM [31]. Similar frameworks have recently been used for minimizing the energy consumption for real-time tasks in systems that allow dynamic voltage and frequency scaling [33].

The schedulability test presented in our paper is based on the utilization of the task set. In order to obtain a test that is not overly pessimistic one must, as discussed before, control Dhall's effect. Schedulability tests that are based on more detailed analysis of the demand and supply times do not need to handle Dhall's in the same explicit way, i.e., in that case one may accept that a task set with very low utilization will miss its deadline as long as the priority assignment algorithm is capable of scheduling more task sets on average (the probability of seeing a specific real-time task set is, however, unclear, and as a consequence of this, it is unclear what it means to be better in the average case). Using similar scenarios as the ones used in Section 6 in this paper, Davis and Burns have in [32] shown that the heuristic priority assignment algorithm DkC [32] performs well on average. DkC is based on a priority assignment algorithm called TkC [35]. In TkC the priority of a task τ_i is based on $T_i - kC_i$ and in DkC the priority of a task τ_i is based on $D_i - kC_i$, where T_i is the period, D_i the deadline, and C_i the worst-case execution time of the task; k is a real number computed as $k = \frac{m-1+\sqrt{5m^2-6m+1}}{2m}$, where m is the number of processors. It can be noted that k is increasing, and that $k = \frac{1+\sqrt{5}}{2} \approx 1.618$, when $m \to \infty$. For both TkC and DkC priority assignment, it is possible to define a task set with arbitrarily low utilization that misses its deadline due to Dhall's effect when we have a multiprocessor with a large number of processors (e.g., by extending the example in [34]).

In [10], the authors studied a reservation-based algorithm, i.e., a Constant Bandwidth Server (CBS) on top of Earliest Deadline First (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 results show that VM technology and the scheduling algorithm affect the real-time application performance. The authors propose to use a less pessimistic analysis to dimension the VM scheduling parameters if one uses the CBS algorithm.

In [11], the authors studied a scheduling method of real-time applications on multicore hardware using a so called synchronized deferrable server. Given a task set, they categorize it into non-migrating tasks and migrating tasks. Non-migrating tasks are statically bounded to a core and are modeled as sporadic tasks whereas migrating tasks can migrate across the cores. This results in using partitioned and global scheduling at the same time.

In [12], the authors presented a method to schedule real-time applications on multiprocessors by virtualizing the processors. Tasks are clustered and each cluster is assigned to a set of virtual processors. In that study, scheduling is done at two levels. The first is scheduling virtual processors onto physical processors and the second is scheduling tasks on a set of virtual processors.

In [13], the authors developed a Compositional Scheduling Architecture (CSA) that is built on the Xen virtualization platform. The architecture allows timing isolation among VMs and supports timing guarantees for real-time tasks. In [14], the authors present a model that includes the cache in hierarchical scheduling while keeping temporal isolation between applications that share a uniprocessor.

In [15], the authors propose a mechanism to schedule soft real-time systems, which provides temporal isolation between VMs that share a processor. In [16], [17], the authors present a model that accounts for the overhead, in compositional hierarchical scheduling for uniprocessors. In [18] compositional scheduling theory was also applied to a multi-core VM scheduler for the Xen real-time virtualization platform.

In connection with hard deadlines systems, the utilization has been studied for real-time systems that respond to 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 [1], 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. The authors considered a uniprocessor VM, i.e., a VM with one virtual core.

In [20], the authors gave a utilization bound of $3-\sqrt{7} \approx 0.35425$ for aperiodic tasks where the utilization threshold is based on the maximum synthetic utilization. If the load on the multiprocessor server stays below 0.35425, the server guarantees the schedulability of all tasks. This bound was derived from the function $u(x) = 2(1-x)/(2+\sqrt{2+2x})$ where the variable x is the utilization of the task with the longest period. The bound is based on a scheduling policy where tasks with a utilization below the bound (light tasks) are scheduled based on earliest deadline first. This bound was later improved to $(3-\sqrt{5})/2 \approx 0.38197$ by scheduling tasks with a utilization below the bound based on least slack first [34].

In [21], the authors proposed an approach to schedule real-time tasks on virtual processors using clustering techniques. Cluster-based scheduling, can be viewed as a generalization of partitioned and global scheduling, where tasks are statically assigned to a cluster and then globally scheduled within the cluster. Cluster-based scheduling can be classified into two types: physical and virtual depending on how a cluster is mapped to the processors in the platform. Physical clusters share no processors in the platform, i.e., a processor in a multiprocessor can only belong to one cluster. Virtual clusters can time-share processors, i.e., a processor in a multiprocessor can belong to one cluster at one time and then belong to another cluster at another time.

In [22], the authors proposed predictable virtual processors for real-time systems where all virtual processors are scheduled on top of the same physical processor using earliest-deadline-first or rate monotonic scheduling. The method proposed by the authors creates the illusion of a dedicated processor for each virtual processor. Real-time tasks are scheduled on top of a virtual processor as if it was a physical processor. Each virtual processor is assigned a utilization that can guarantee the schedulability of the real-time tasks that are running on that processor. In [23], the authors presented a methodology for running periodic and sporadic real-time applications with relative deadlines that share resources on a multiprocessor platform; the physical multiprocessor is abstracted through sets of virtual processors.

5.3 Definition

We consider the problem of scheduling a task set $\Gamma = \{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$ consisting of n independent, periodic real-time tasks on m identical virtual processors in a VM that runs a time C_{VM} periodically every T_{VM} time interval with a blocking time of Bp ($Bp = T_{VM} - C_{VM}$) during which the VM does not have access on the hardware. An example of a periodic multiprocessor VM is Xen [24]. We also assume that there is a deadline D_{VM} associated with the VM. The VM needs to finish its execution time C_{VM} within D_{VM} time units after its release; we assume that $C_{VM} \leq D_{VM} \leq T_{VM}$, i.e., constrained deadlines. We assume that m physical processors are available to the VM during the time interval C_{VM} in every period (see Figure 1).

The multiprocessor VM schedules the runnable tasks by arranging them in a shared ready queue and dequeues them in priority order as soon as a (virtual) processor becomes available (as always in virtualized systems, each VM is unaware of the fact that it does not have constant access to the underlying hardware). We assume preemptive scheduling, i.e., a lower priority task may be preempted when a higher priority task becomes runnable. The preempted task is in that case put back into the ready queue, and can later be restarted on any available processor. We assume that there is no overhead or other penalty for switching from one task to another. Each task can execute on at most one processor at a time.

The problem is thus defined by five parameters:

- $\Gamma = \{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$
- m
- Bp ($Bp = T_{VM} - C_{VM}$), i.e., $C_{VM} = T_{VM} - Bp$
- T_{VM}
- D_{VM}

Based on these five parameters we want to determine:

- If we can guarantee that $\Gamma = \{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$ is schedulable under these conditions.
- How we should assign priorities to the n tasks in Γ in case we can guarantee that Γ is schedulable.

Tasks can be either heavy or light (based on their utilization). Heavy tasks get higher priority than light tasks, and light tasks get priorities according to the Rate Monotonic (RM) priority assignment algorithm. RM assigns priorities to tasks according to their periods; tasks with shorter periods get higher priorities.

In order to solve the problem defined above, we start by considering a task set X that consists of a task $\tau_x = (C_x, T_x)$ and an arbitrary number of tasks with higher priority, and thus shorter periods, than τ_x (at the moment we assume that all tasks are light). We consider critical task sets, which means that τ_x meets its deadlines in all periods, but any extension of C_x results in a missed deadline for τ_x. Let X' denote task set X when we remove τ_x. It is clear that for a certain task τ_x there is an infinite number of task sets X'. Let $\overline{X'}$ denote the set of all possible task sets X' for a given τ_x and let X'' denote a task set in $\overline{X'}$ with minimum utilization, and let U'' denote the utilization of task set X''. We refer to X'' as an extremal task set; for a certain task τ_x there could be many extremal task sets, all these will, however, have the same utilization U''.

Theorem 1: For $Bp = 0$, consider a task set X_e that consists of km tasks

where all tasks have $C = e$, and where k is the number of task that run on a single virtual processor and m is the number of virtual processors. $\tau_{1,1} = (e, T), \tau_{1,2} = (e, T), \ldots, \tau_{1,m} = (e, T)$
$\tau_{2,1} = (e, T + e), \tau_{2,2} = (e, T + e), \ldots, \tau_{2,m} = (e, T + e)$
$\tau_{3,1} = (e, T + 2e), \tau_{3,2} = (e, T + 2e), \ldots, \tau_{3,m} = (e, T + 2e)$
.
.
.
$\tau_{k,1} = (e, T + (k-1)e), \tau_{k,2} = (e, T + (k-1)e), \ldots, \tau_{k,m} = (e, T + (k-1)e)$,
where $T = \frac{T_x + C_x}{2}$, and $ke = T_x - T$.
As $e \to 0$ and $k \to \infty$ while preserving all equations, the task set X_e becomes liquid (by definition) and also extremal, with $X_e \to X''$
Proof: When $Bp = 0$ there is no virtualization, and it is therefore the same case as in [2].

Figure 3 shows how this task set executes on a number of identical virtual processors, i.e., $m = 3$; shaded areas indicate the time periods when the task set is executing.

Corollary 1: In an extremal and liquid task set, a task τ_x should be classified as a light task if $U'' \leq ln\left(1 + \frac{d}{T}\right)$ otherwise it should be classified as a heavy task. U'' is the utilization of the task set X_e in Theorem 1.

Proof: Let k denote the number of execution slots and d denote the length of the interval occupied by the k slots. The total utilization of the tasks $\tau_{1,1}, \ldots \tau_{k,1}$ defined in Theorem 1 is $\Sigma_{i=1}^{k} \frac{e}{T+(i-1)e}$. If $e \to 0$ and $k \to \infty$ so

that $ke = d$ is constant, the utilization U'' is a decreasing function of m [25]. We thus obtain a lower bound after this limit, in which a Riemann sum converges to an integral:

$$\Sigma_{i=1}^{k} \frac{e}{T+(i-1)e} \to \int_0^d \frac{dt}{T+t} = ln\left(1 + \frac{d}{T}\right). \qquad (4) \ ■$$

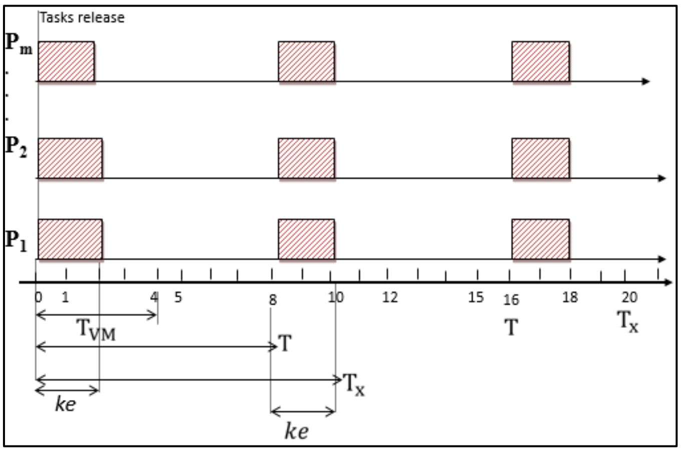


Figure 3. Scheduling mk tasks on m processor when $Bp = 0$.

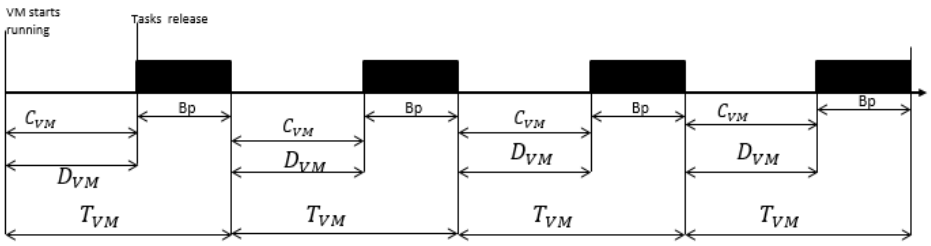


Figure 4. Scheduling a VM with $D_{VM} = T_{VM} - Bp$

When $Bp > 0$, we can construct a task set X'' similar to when $Bp = 0$. We must make sure that all processors are idle at time $T_x + e$ (which we had for the task set described in Theorem 1 when $Bp = 0$). It is also clear that in an extremal task set X'', the second arrival of any task in the interval [0, T_x] should not occur during a period when the VM does not have access to the hardware. The reason for this is that if a task τ_i from X'' arrives when the VM does not have access to the hardware, the period T_i of task τ_i can be extended without reducing the interference on τ_x, since all tasks in X'' execute two times before time T_x. This is a contradiction to the definition of extremal task sets, since the utilization of X'' will decrease if we extend some period. Obviously, tasks can only execute during periods when the VM has access to the hardware.

In the rest of this section we assume that $D_{VM} = C_{VM}$, i.e., constrained deadlines. As a consequence, the blocking periods *Bp* come at the same place in all periods T_{VM}. In the worst-case scenario, the task set is released at the start of a time period when the VM does not have access to the hardware. As a consequence, the tasks start executing in the VM after a blocking time Bp (see Figure 4). In Section 5 we relax this restriction and look at the general case when we can have different values of D_{VM}. However, we always assume that $D_{VM} \leq T_{VM}$.

Theorem 2: Consider four integers $a_k, b_k, c_k, z \geq 0$, where a_k, b_k, c_k denote the number of execution slots of length e in three types of intervals; and let a, b, c denote the length of the intervals that these slots block, and let *z* be the number of intervals of length b. When $Bp > 0$ and $D_{VM} = T_{VM} - Bp$, there is a task set X_e'' that consists of $a_k m + z b_k m + c_k m$ tasks such that

$\tau_{1,1} = \tau_{1,2} =, \dots, = \tau_{1,m} = (e, T)$

$\tau_{2,1} = \tau_{2,2} =, \dots, = \tau_{2,m} = (e, T + e)$

.

.

.

$\tau_{a_k,1} = \tau_{a_k,2} =, \dots, = \tau_{a_k,m} = (e, T + (a_k - 1)e)$

$\tau_{a_k+1,1} = \tau_{a_k+1,2} =, \dots, = \tau_{a_k+1,m} = (e, T + Bp + a_k e)$

$\tau_{a_k+2,1} = \tau_{a_k+2,2} =, \dots, = \tau_{a_k+2,m} = (e, T + Bp + a_k e + e)$.

.

.

$\tau_{a_k+b_k,1} = \tau_{a_k+b_k,2} =, \dots, = \tau_{a_k+b_k,m} = (e, T + Bp + a_k e + (b_k - 1)e)$

$\tau_{a_k+b_k+1,1} = \tau_{a_k+b_k+1,2} =, \dots, = \tau_{a_k+b_k+1,m}$
$= (e, T + 2Bp + a_k e + b_k e)$

$\tau_{a_k+b_k+2,1} = \tau_{a_k+b_k+2,2} =, \dots, = \tau_{a_k+b_k+2,m}$
$= (e, T + 2Bp + a_k e + b_k e + e)$

.

.

.

$\tau_{a_k+2b_k,1} = \tau_{a_k+2b_k,2} =, \dots, = \tau_{a_k+2b_k,m}$
$= (e, T + 2Bp + a_k e + b_k e + (b_k - 1)e)$

.

.

.

$\tau_{a_k+(z-1)b_k+1,1} = \tau_{a_k+(z-1)b_k+1,2} =, \dots, = \tau_{a_k+(z-1)b_k+1,m} =$
$(e, T + zBp + a_k e + (z - 1)b_k e)$

$\tau_{a_k+(z-1)b_k+2,1} = \tau_{a_k+(z-1)b_k+2,2} =, \dots, = \tau_{a_k+(z-1)b_k+2,m} =$
$(e, T + zBp + a_k e + (z-1)b_k e + e)$

.

.

.

$\tau_{a_k+zb_k,1} = \tau_{a_k+zb_k,2} =, \dots, = \tau_{a_k+zb_k,m} =$
$(e, T + zBp + a_k e + (z-1)b_k e + (b_k - 1)e)$

$\tau_{a_k+zb_k+1,1} = \tau_{a_k+zb_k+1,2} =, \dots, = \tau_{a_k+zb_k+1,m} =$
$(e, T + (z+1)Bp + a_k e + zb_k e)$

$\tau_{a_k+zb_k+2,1} = \tau_{a_k+zb_k+2,2} =, \dots, = \tau_{a_k+zb_k+2,m} =$
$(e, T + (z+1)Bp + a_k e + zb_k e + e)$

.

.

.

$\tau_{a_k+zb_k+c_k,1} = \tau_{a_k+zb_k+c_k,2} =, \dots, = \tau_{a_k+zb_k+c_k,m} =$
$(e, T + (z+1)Bp \ + a_k e + zb_k e + (c_k - 1)e)$

When $e \to 0$ and $a_k, b_k, c_k \to \infty$ with $a_k e = a, b_k e = b,$ and $c_k e = c$ we obtain $X_e'' \to X''$ and minimal utilization: $U(X'') \leq U(X_e'')$. Period T represents the period for liquid tasks, in order to be able to schedule our tasks as if they are running on a physical uniprocessor, we embed the virtualization properties such as T_{VM} and *Bp* into *T*.

$$T = \frac{T_x - \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor Bp - min\left(T_x - T_{VM} \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor, Bp \right) + C_x}{2} + \left(\left\lfloor \frac{\left(T_x - \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor Bp - min\left(T_x - T_{VM} \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor, Bp \right) + C_x \right) / 2}{T_{VM} - Bp} \right\rfloor + 1 \right) Bp \qquad (5)$$

$$a = min\left(T_x, T_{VM} \left\lceil T / T_{VM} \right\rceil \right) - T \qquad (6)$$

If $a = T_x - T$ then $z = 0$, $b = 0$, and $c = 0$ (see Figure 5).

$$z = max(0, \lfloor T_x / T_{VM} \rfloor - \lceil T / T_{VM} \rceil) \qquad (7)$$

If $z > 0$ then $b = T_{VM} - Bp$ (8)

$$c = max(0, T_x - \lfloor T_x / T_{VM} \rfloor T_{VM} - Bp) \qquad (9)$$

Proof: The available time (non-blocked time) T_a, during which task τ_x has to complete its execution, is T_x minus the time when the VM is blocked. The available time T_a is thus

$$T_a = T_x - \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor Bp - min\left(T_x - T_{VM} \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor, Bp \right). \tag{10}$$

The minimum depends on whether T_x is in a blocked interval or not.

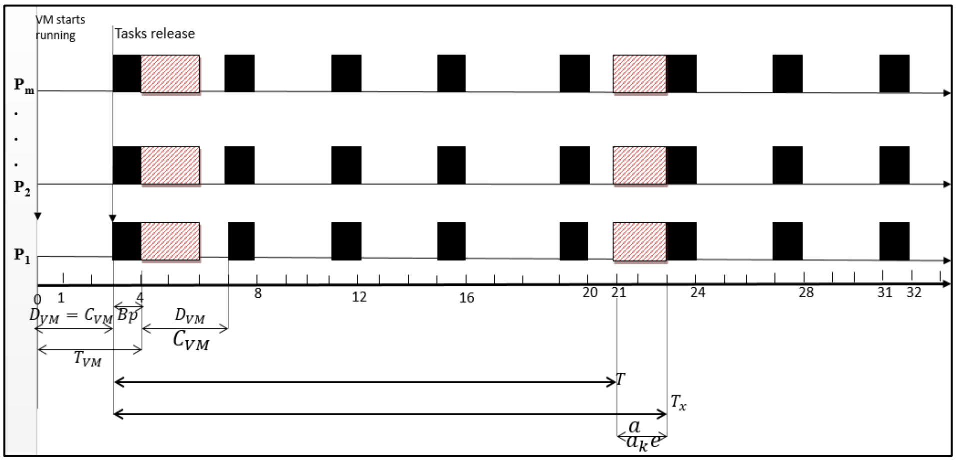


Figure 5. Scheduling $m * a_k$ tasks on m virtual processors when $Bp > 0$, $a = T_x - T$, $z = 0$, and $c = 0$

From the definition of X'' we know that in the interval $[0, T_x]$ this task set will generate interference on τ_x that will keep all processors busy during a total time interval of $T_a - C_x$. Half of this interference is generated by the first release of each task and the other half is generated by the second release of each task. The available time before time T is $T - \left(\left\lfloor \frac{T}{T_{VM}} \right\rfloor + 1 \right) Bp$, since time T is always in a period when the VM has access to the processors. Therefore we obtain Equation 11 that covers all the time that VM has access to the hardware during time interval [0, T_x].

$$C_x + 2\left(T_a - \left(T - \left(\left\lfloor \frac{T}{T_{VM}} \right\rfloor + 1 \right) Bp \right) \right) = T_a \tag{11}$$

From Equation 11,

$$T - \left(\left\lfloor \frac{T}{T_{VM}} \right\rfloor + 1 \right) Bp = \frac{T_a + C_x}{2} \tag{12}$$

We want to find a solution T, this solution turns out to be

$$T = \frac{(T_a + C_x)}{2} + \left(\left\lfloor \frac{(T_a + C_x)/2}{\mathrm{T}_{\mathrm{VM}} - Bp} \right\rfloor + 1 \right) Bp \qquad (13)$$

If we replace T_a in Equation 13 by its value (see Equation 10) we get T,

$$T = \frac{T_x - \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor Bp - min\left(T_x - T_{VM} \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor, Bp \right) + C_x}{2}$$
$$+ \left(\left\lfloor \frac{\left(T_x - \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor Bp - min\left(T_x - T_{VM} \left\lfloor \frac{T_x}{T_{VM}} \right\rfloor, Bp \right) + C_x \right) / 2}{T_{VM} - Bp} \right\rfloor + 1 \right) Bp \qquad \blacksquare$$

The period of τ_x may either end during a period when the VM is blocked (in this case $c = 0$ in Theorem 2), or in a period when the VM has access to the processors (in this case $c > 0$, or $a = T_x - T$ in Theorem 2). If the period of τ_x ends during a period when the VM is blocked, we have two cases: $z = 0$ or $z > 0$. If $z = 0$ we only have the *a*-component of the interference (see Figure 5). If $z > 0$, we have one or more *b*-components (see Figure 7). It is clear that the task set in Theorem 2 fills each *b*-component using minimum utilization of the interfering tasks. Also, the *a*-component is filled using minimum utilization in the same way as in Theorem 1. If the period of τ_x ends during a period when the VM has access to the processors, we have three cases: $c = 0$ and $z = 0$, $c > 0$ and $z = 0$, or $c > 0$ and $z > 0$. If $c = 0$ and $z = 0$, we only have the *a*-component of the interference (see Figure 4). In this case we have the same situation as in Theorem 1. If $c > 0$ and $z = 0$, we have no *b*-component (see Figure 6).

It is clear that the task set in Theorem 2 fills the *a*-component and the *c*-component using minimum utilization of the interfering tasks. If $c > 0$ and $z > 0$ we have the situation shown in Figure 8. It is again clear that the task set in Theorem 2 fills the *a*-, *b*-, and *c*- components using minimum utilization of the interfering tasks.

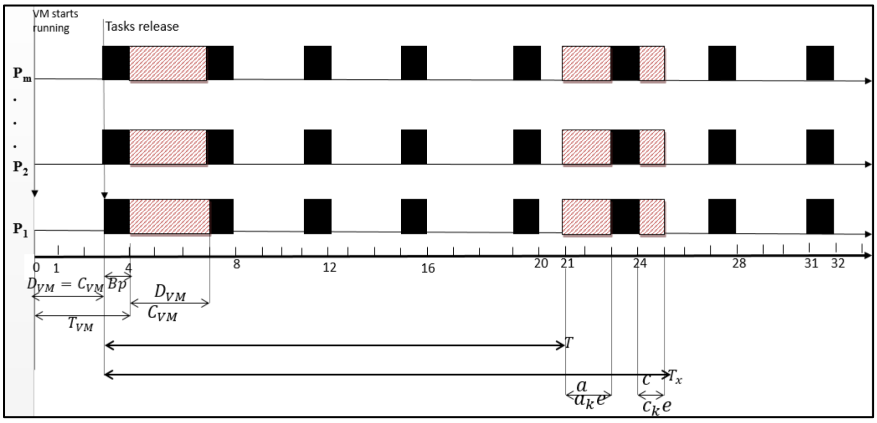


Figure 6. Scheduling $m(a_k + c_k)$ tasks on m processors when $Bp > 0$, $a = \left(T_{VM}\left\lceil {}^{T}/_{T_{VM}}\right\rceil\right) - T$, $z = 0$ and $c > 0$

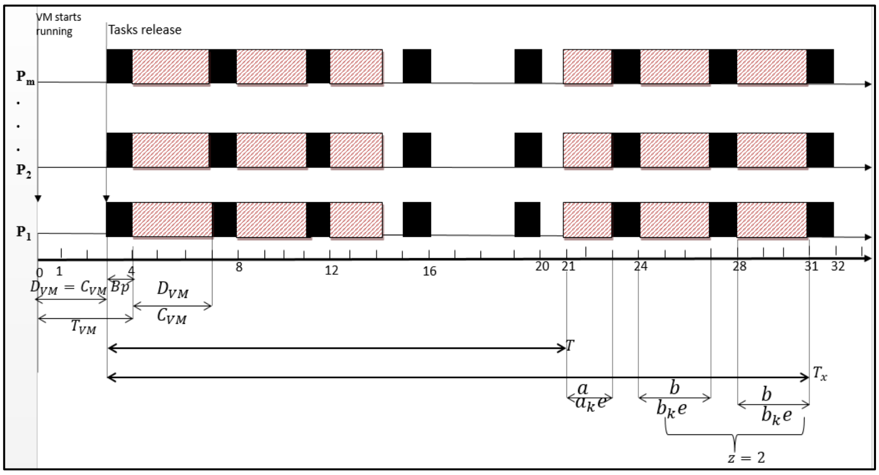


Figure 7. Scheduling $m(a_k + 2b_k)$ tasks on m processor when $Bp > 0$, $a = \left(T_{VM}\left\lceil {}^{T}/_{T_{VM}}\right\rceil\right) - T$, $z > 0$ and $c = 0$

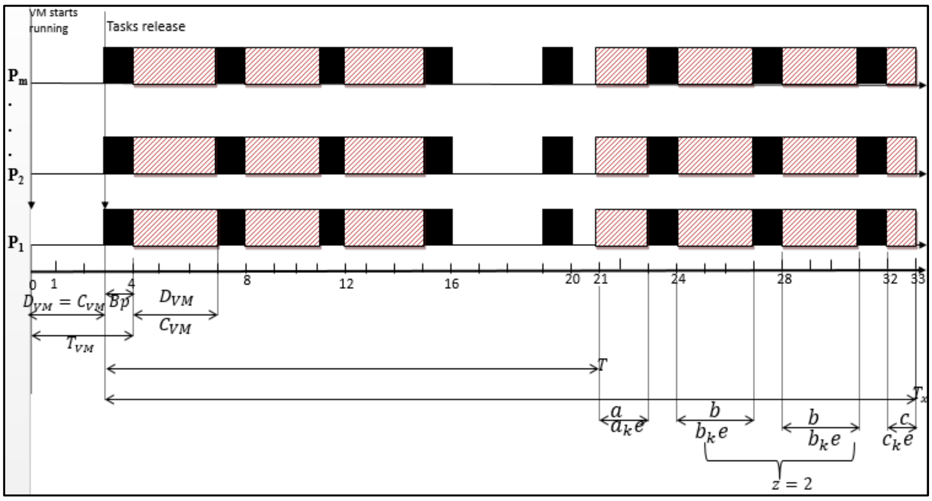


Figure 8. Scheduling $m(a_k + 2b_k + c_k)$ tasks on *m* processor when $Bp > 0$, $a = \left(T_{VM} \left\lceil {}^{T}/_{T_{VM}} \right\rceil\right) - T$, $z > 0$ and $c > 0$.

Corollary 2: In an extremal and liquid task set, a task τ_x should be classified as a light task if Equation 14 is true, otherwise it should be classified as a heavy task and has higher priority than light tasks.

$$U'' \leq ln\left(1 + \frac{a}{T}\right) + \sum_{s=1}^{s=z} ln\left(1 + \frac{b}{T+a+sBp+(s-1)b}\right) + ln\left(1 + \frac{c}{T+a+(z+1)Bp+zb}\right) \tag{14}$$

where U'' is the utilization of the extremal task set X''.

Proof: The total utilization of the tasks $\tau_{1,1}$, …,$\tau_{a_k,1}$ defined in Theorem 2 equals $\Sigma_{i=1}^{a_k} \frac{e}{T+(i-1)e}$. If $e \to 0$ and $a_k \to \infty$ so that $a_k e = a$ is constant, the utilization $\Sigma_{i=1}^{a_k} \frac{e}{T+(i-1)e}$ is a decreasing function of m [25]. We thus obtain a lower bound after this limit, in which a Riemann sum converges to an integral:

$$\Sigma_{i=1}^{a_k} \frac{e}{T+(i-1)e} \to \int_0^a \frac{dt}{T+t} = ln\left(1 + \frac{a}{T}\right) \tag{15}$$

Corollary 2 follows by using a similar argument for the other parts of the task set, i.e., parts b and c. ■

5.4 Algorithm Description

The following is the description of the algorithm that is shown in Figure 9.

1. *Given a task set* $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$,
2. $i = 0$
3. $i \coloneqq i + 1$
4. *Given*

$$T = \frac{T_i - \left\lfloor\frac{T_i}{T_{VM}}\right\rfloor Bp - min\left(T_i - T_{VM}\left\lfloor\frac{T_i}{T_{VM}}\right\rfloor, Bp\right) + C_i}{2} + \left(\left\lfloor\frac{\left(T_i - \left\lfloor\frac{T_i}{T_{VM}}\right\rfloor Bp - min\left(T_i - T_{VM}\left\lfloor\frac{T_i}{T_{VM}}\right\rfloor, Bp\right) + C_i\right)\Big/2}{T_{VM} - Bp}\right\rfloor + 1\right) Bp$$

If $\frac{1}{m}\sum_{j=1}^{i-1} C_j/T_j \geq ln\left(1 + \frac{a}{T}\right) + \sum_{s=1}^{z} ln\left(1 + \frac{b}{T+a+sBp+(s-1)b}\right) + ln\left(1 + \frac{c}{T+a+(z+1)Bp+zb}\right)$, *then* τ_i *is a light task,* do $i \coloneqq i + 1$

5. *repeat until i==n,*
6. $\mathit{\Gamma}$ *is schedulable, end*
   else τ_i *is a heavy task, continue to Step 7.*

7. *Give task* τ_i *the highest priority, // we mark it as heavy. (N.B. we cannot have more than m heavy tasks).*
8. *if* $C_i \leq T_i - \left\lfloor\frac{T_i}{T_{VM}}\right\rfloor Bp - min\left(T_i - T_{VM}\left\lfloor\frac{T_i}{T_{VM}}\right\rfloor, Bp\right)$ *(16)*
9. *and* $i == n$ *and* $i == m$

then $\mathit{\Gamma}$ *is schedulable, end. //(this is a special case that can only happen when* $n == m$*)*

10. *elseif m > 1 then*
11. $(m \coloneqq m - 1), \Gamma \coloneqq \Gamma \backslash \{\tau_i\}$, *reset* $i = 0$,

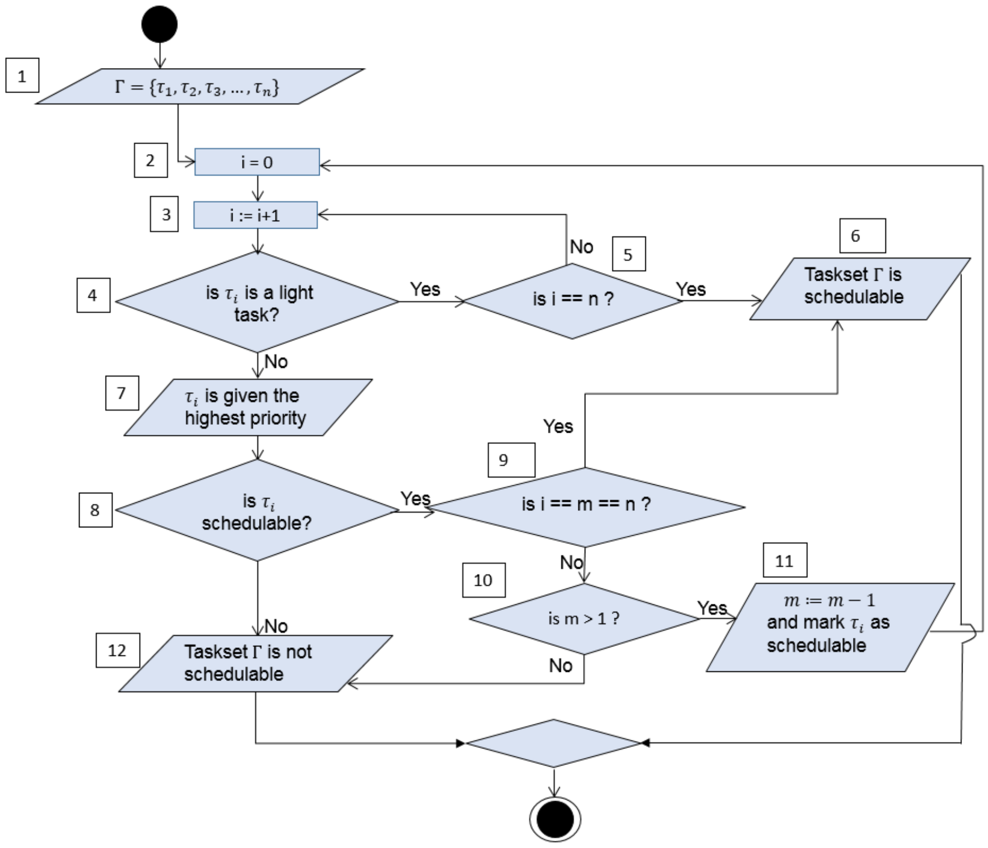
else

12. $\mathit{\Gamma}$ *is not schedulable, end*

We can have at most n "Yes" alternatives in a row in Step 4 (the inner loop), and we can have at most m "No" alternatives in Step 9 (the outer loop).

The total number of loops equals to:

$n + (n-1) + (n-2) + \cdots + \big(n - (m-1)\big) = mn - \big(1 + 2 + \cdots + (m-1)\big) = mn - \left(\frac{(m-1)m}{2}\right) = mn\left(1 - \left(\frac{(m-1)}{2n}\right)\right)$. The complexity of the algorithm is thus $O(mn)$.



1
$\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$
2
i = 0
3
i := i+1
4
is τ_i is a light task?
Yes
No
5
is i == n ?
Yes
6
Taskset Γ is schedulable
No
7
τ_i is given the highest priority
Yes
9
8
is τ_i schedulable?
Yes
is i == m == n ?
No
10
11
is m > 1 ?
Yes
$m := m - 1$ and mark τ_i as schedulable
No
12
Taskset Γ is not schedulable
No

Figure 9. Algorithm flowchart

5.4.1 Example

We consider the task set $\Gamma = \{\tau_1, \tau_2, \tau_3, \dots, \tau_8\}$
$\tau_1, C_1 = 2, T_1$=12; $\tau_2, C_2 = 2, T_2$ =12; $\tau_3, C_3 = 2, T_3$ =12; $\tau_4, C_4 = 2, T_4$ =12; $\tau_5, C_5 = 2, T_5$ =12;
$\tau_6, C_6 = 2,\ T_6$ =12; $\tau_7,\ C_7 = 2,\ T_7$ =12; $\tau_8, C_8 = 10, T_8 = 33.$

Consider a VM with $m = 3, T_{VM} = 6$ and $Bp = 2$. The critical task is τ_8, in order to save space we skip the analysis of the first seven tasks. By inserting $T_x = T_8 = 33, T_{VM} = 6,\ Bp = 2$, and $C_x = C_8 = 10$ in

$$T = \frac{T_x - \left\lfloor\frac{T_x}{T_{VM}}\right\rfloor Bp - min\left(T_x - T_{VM}\left\lfloor\frac{T_x}{T_{VM}}\right\rfloor, Bp\right) + C_x}{2}$$
$$+\left(\left\lfloor\frac{\left(T_x - \left\lfloor\frac{T_x}{T_{VM}}\right\rfloor Bp - min\left(T_x - T_{VM}\left\lfloor\frac{T_x}{T_{VM}}\right\rfloor, Bp\right) + C_x\right)/_2}{T_{VM} - Bp}\right\rfloor + 1\right)Bp = 23.5$$

The utilization U'' of τ_1 to τ_7 equals $\frac{1}{3}\sum_1^7 \frac{2}{10} \approx 0.467$

$$a = min\left(33\,, 6\left\lceil\frac{23.5}{6}\right\rceil\right) - 23.5 = 0.5$$
$$z = max\left(0, \left\lfloor\frac{33}{6}\right\rfloor - \left\lceil\frac{23.5}{6}\right\rceil\right) = 1$$
$$b = 6 - 2 = 4$$
$$c = max\left(0, 33 - \left\lfloor\frac{33}{6}\right\rfloor 6 - 2\right) = 1$$

From Equation 14, by replacing *T*, *a*, *b*, *c*, and *z*, the threshold equals to

$$ln\left(1 + \frac{0.5}{23.5}\right) + \sum_{s=1}^{1} ln\left(1 + \frac{4}{23.5 + 1 + 2 + (1 - 1)4}\right)$$
$$+ ln\left(1 + \frac{1}{23.5 + 1 + (1 + 1)2 + (1 * 4)}\right)$$
$$= ln\left(1 + \frac{0.5}{23.5}\right) + ln\left(1 + \frac{4}{26.5}\right) + ln\left(1 + \frac{1}{32.5}\right) \approx 0.192$$

Since the threshold is smaller than U'', i.e., $0.192 < 0.467$, we allocate τ_8 to its own processor and check if it is schedulable (Step 4 in the algorithm). Since $10 \le 33 - \left\lfloor\frac{33}{6}\right\rfloor 2 - min\left(33 - 6\left\lfloor\frac{33}{6}\right\rfloor, 2\right)$, i.e., $10 \le 21$, τ_8 is schedulable. The next step is to check one by one if the remaining tasks $\tau_1, \tau_2, \dots \tau_7$ are schedulable on the remaining two processors. It turns out that the task set is schedulable. Figure 10 shows that this task set is not schedulable using rate monotonic priority assignment. However, it is schedulable when we use the heavy and light classification (see Figure 11). In Figure 11, the heavy task τ_8 executes uninterruptedly, as if it had a processor of its own.

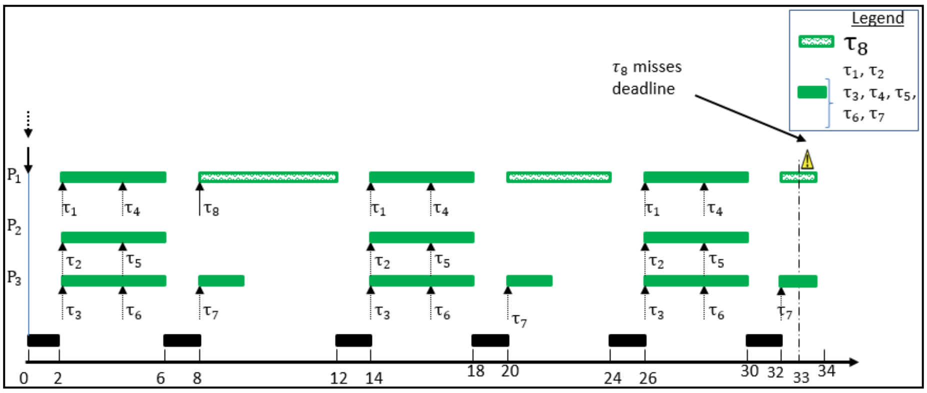


Figure 10. Scheduling a task set of 8 tasks on a VM with $m = 3$, $T_{VM} = 6$, $Bp = 2$ using RM

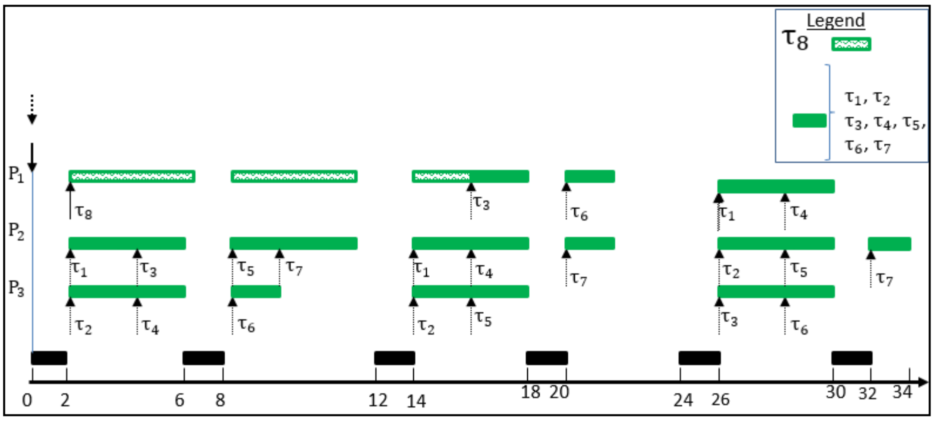


Figure 11. Scheduling a task set of 8 tasks on a VM with $m = 3$, $T_{VM} = 6$ $Bp = 2$ when classifying τ_8 as heavy

5.5 The effect of different deadlines (D_{VM})

Large values on D_{VM} increase the possible variation of the position of the blocking period *Bp* during each period T_{VM}. The longest time that a VM is blocked occurs when the VM execution time (C_{VM}) comes as early as possible in one period (i.e., immediately after the release of the VM, thus making *Bp* occur at the end of the T_{VM} period), and the VM execution time (C_{VM}) comes as late as possible in the next periods. In that case the VM will be blocked during a time period of *Bp* + D_{VM} - C_{VM}, e.g., if $D_{VM} = T_{VM}$ the VM will be blocked during a period of length 2*Bp*. Figure 12 shows how the VM is executed in the worst case scenario for $D_{VM} = T_{VM}$. The extreme case with two *Bps* consecutively, is shown in Figure 13 for a multiprocessor VM with three virtual processors. A similar periodic VM is discussed in [1]. Figures 14 and 15 show the corresponding worst case scenarios for a D_{VM} longer than C_{VM} but shorter than T_{VM}. As a result, Theorem 3 below replaces Theorem 2 in Step 4 in the algorithm described in Section 4.

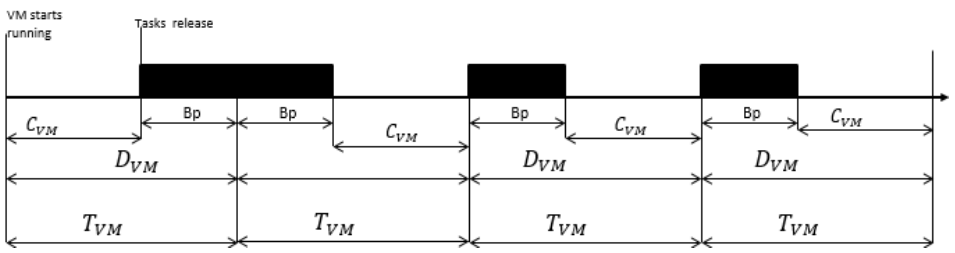


Figure 12. Worst-case scenario for $D_{VM} = T_{VM}$

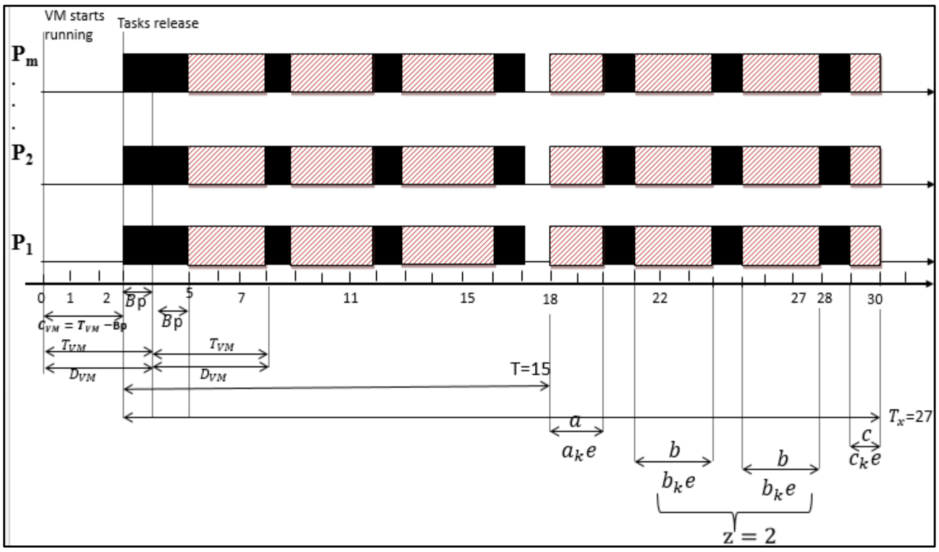


Figure 13. The worst case scenario: Scheduling $m(a_k + 2b_k + c_k)$ tasks on m processor when $Bp > 0$, $a = \left(T_{VM}\left\lceil T/T_{VM}\right\rceil\right) - T$, $z > 0$ and $c > 0$, with $D_{VM} = T_{VM}$

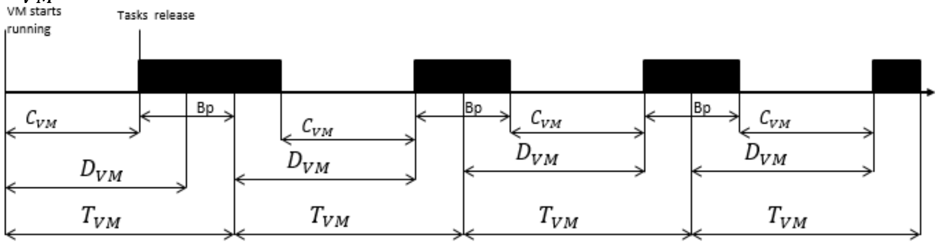


Figure 14. Scheduling VM with $T_{VM} - Bp < D_{VM} < T_{VM}$.

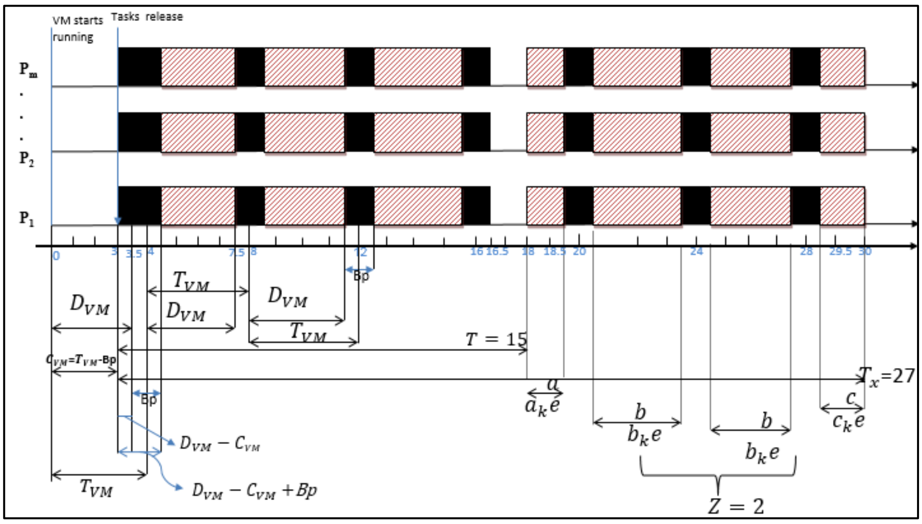


Figure 15. The worst case scenario: Scheduling $m(a_k + 2b_k + c_k)$ tasks on m processor when $Bp > 0$, $a = \left(T_{VM} \left\lceil {}^{T}/_{T_{VM}} \right\rceil\right) - T$, $z > 0$ and $c > 0$. With $T_{VM} - Bp \leq D_{VM} \leq T_{VM}$

The period T is affected by D_{VM}, the formula in Equation 5 that is used in the algorithm in Step 4, is for the general case computed using Equation 18 in Theorem 3 (see below). Step 8 in the algorithm also becomes Equation 17 to determine if a heavy task τ_i is schedulable on its own dedicated processor.

$$C_i \leq T_i - D_{VM} + T_{VM} - Bp - \left\lfloor \frac{T_i - D_{VM} + T_{VM} - Bp}{T_{VM}} \right\rfloor Bp - min\left(T_i - D_{VM} + T_{VM} - Bp - T_{VM} \left\lfloor \frac{T_i - D_{VM} + T_{VM} - Bp}{T_{VM}} \right\rfloor, Bp \right). \quad (17)$$

Theorem 3: We use the same task set as in Theorem 2. When we consider D_{VM} the formulas for T, a, b, c, and z are as follows:

$$T = \frac{\left(\begin{array}{c} T_x - D_{VM} + T_{VM} - Bp - \left\lfloor \frac{T_x - D_{VM} + T_{VM} - Bp}{T_{VM}} \right\rfloor Bp \\ - min\left(T_x - D_{VM} + T_{VM} - Bp - T_{VM} \left\lfloor \frac{T_x - D_{VM} + T_{VM} - Bp}{T_{VM}} \right\rfloor, Bp \right) + C_x \end{array} \right)}{2}$$

$$+\left(\left\lfloor\frac{\left(\begin{matrix}T_x-D_{VM}+T_{VM}-Bp-\left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor Bp\\-min\left(T_x-D_{VM}+T_{VM}-Bp-T_{VM}\left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor,Bp\right)+C_x\end{matrix}\right)\Big/2}{T_{VM}-Bp}\right\rfloor+1\right)Bp+(D_{VM}-T_{VM}+Bp)$$

(18)

$a = min\left(T_x - D_{VM} + T_{VM} - Bp, T_{VM}\left\lceil\frac{T-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rceil\right) - (T - D_{VM} + T_{VM} - Bp)$ (19)

If $a = T_x - T$ then $z = 0$, $b = 0$, and $c = 0$

$z = max\left(0, \left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor - \left\lceil\frac{T-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rceil\right)$ (20)

if $z > 0$ then $b = T_{VM} - Bp$ (21)

$c = max(0, T_x - D_{VM} + T_{VM} - Bp - \left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor T_{VM} - Bp)$ (22)

Proof: The available time (non-blocked time) that task τ_x has to complete its execution is T_x minus the time when the VM is blocked. The available time denoted by T_a is thus (see Figure 15):

$T_a = T_x - (D_{VM} - C_{VM}) - \left\lfloor\frac{T_x-(D_{VM}-C_{VM})}{T_{VM}}\right\rfloor Bp - min\left(T_x - (D_{VM} - C_{VM}) - T_{VM}\left\lfloor\frac{T_x-(D_{VM}-C_{VM})}{T_{VM}}\right\rfloor, Bp\right)$ (23)

with $C_{VM} = T_{VM} - Bp$ we get

$T_a = T_x - D_{VM} + T_{VM} - Bp - \left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor Bp - min\left(T_x - D_{VM} + T_{VM} - Bp - T_{VM}\left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor, Bp\right)$ (24)

The minimum depends on whether T_x ends in a blocked interval or not.
From the definition of X'' we know that in the interval [0, T_x] this task set will generate interference that will keep all processors busy during a total time interval of $T_a - C_x$. Half of this interference is generated by the first release of each task and the other half is generated by the second release of each task. The available time before T that the VM has access to the processors is $T - D_{VM} + T_{VM} - Bp - \left(\left\lfloor\frac{T-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor + 1\right)Bp$. This means that the time that τ_x is blocked due to the second release of the tasks in X'' is $T_a -$

$\left(T - D_{VM} + T_{VM} - Bp - \left(\left\lfloor\frac{T-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor + 1\right) Bp\right)$ and since the time that τ_x is blocked due to the first release of the tasks in X'' is the same. Therefore, we get

$$C_x + 2\left(T_a - \left(T - D_{VM} + T_{VM} - Bp - \left(\left\lfloor\frac{T-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor + 1\right) Bp\right)\right) = T_a \tag{25}$$

From (25) we get (26)

$$T - D_{VM} + T_{VM} - Bp - \left(\left\lfloor\frac{T-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor + 1\right) Bp = (T_a + C_x)/2 \tag{26}$$

From (26) we find a solution for T in (27)

$$T = \frac{\begin{pmatrix} T_x - D_{VM} + T_{VM} - Bp - \left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor Bp \\ - min\left(T_x - D_{VM} + T_{VM} - Bp - T_{VM}\left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor, Bp\right) + C_x \end{pmatrix}}{2} + \left(\left\lfloor\frac{\begin{pmatrix} T_x - D_{VM} + T_{VM} - Bp - \left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor Bp \\ - min\left(T_x - D_{VM} + T_{VM} - Bp - T_{VM}\left\lfloor\frac{T_x-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor, Bp\right) + C_x \end{pmatrix}\Big/ 2}{T_{VM} - Bp}\right\rfloor + 1\right) Bp + D_{VM} - T_{VM} + Bp \tag{27}$$ ■

Corollary 3: In an extremal and liquid task set, a task τ_x should be classified as a light task if Equation 28 is true, otherwise it should be classified as a heavy task thus having higher priority than light tasks.

$$U'' \leq ln\left(1 + \frac{a}{T}\right) + \sum_{s=1}^{s=z} ln\left(1 + \frac{b}{T + a + sBp + (s-1)b}\right) + ln\left(1 + \frac{c}{T + a + (z+1)Bp + zb}\right) \tag{28}$$

Proof: Refer to proof of Corollary 2. In Equation 28 the *T a*, *b*, *c*, and *z* parameters are calculated from Equations 18-22.

5.5.1 Example

Consider the example in Section 4.1, and assume that $D_{VM} = T_{VM}$, then using Equation 27, we have $T = 25$.

The utilization $U''$of τ_1 to $\tau_7 = \frac{1}{3}\sum_1^7 \frac{2}{10} \approx 0.467$
$a = min\left(33-2\,, 6\left\lceil\frac{25-2}{6}\right\rceil\right) - (25-2) = 4$, $z = max\left(0, \left\lfloor\frac{33-2}{6}\right\rfloor - \left\lceil\frac{25-2}{6}\right\rceil\right) = 1$, therefore $b = 5$

$$c = max\left(0, 33-2-\left\lfloor\frac{33-2}{6}\right\rfloor 6-2\right) = 0$$

From Corollary 3, Equation 28 and by calculating *T, a, b, c,* and *z* the threshold equals $ln\left(1+\frac{4}{25}\right) + ln\left(1+\frac{5}{25+4+2}\right) \approx 0.298$
Since the threshold is smaller than U'', i.e., $0.298 < 0.467$, we allocate τ_8 to its own processor and then check if it is schedulable (Step 4 in the algorithm).
Since $C_8 \le T_8 - D_{VM} + T_{VM} - Bp - \left\lfloor\frac{T_8-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor Bp - min\left(T_8 - D_{VM} + T_{VM} - Bp - T_{VM}\left\lfloor\frac{T_8-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor, Bp\right)$,

i.e., $10 < 20$, we see that τ_8 is schedulable on its own processor. The next step is to check one by one if the remaining tasks $\tau_1, \tau_2, \ldots \tau_7$ are schedulable on the remaining two processors using RM. τ_7 misses deadline (see Figure 16), hence the task set is not schedulable we need to increase the resources by decreasing the blocking time.

Let us decrease the blocking time Bp, and then schedule the same task set using $T_{VM} = 6,\ Bp = 1$.
Using Equation 27 we get $T = 23$

$$a = min\left(33-1\,, 6\left\lceil\frac{23-1}{6}\right\rceil\right) - (23-1) = 2$$

$$z = max\left(0, \left\lfloor\frac{33-1}{6}\right\rfloor - \left\lceil\frac{23-1}{6}\right\rceil\right) = 1$$

Therefore $b = 6-1 = 5$

$$c = max\left(0, 33-1-\left\lfloor\frac{33-1}{6}\right\rfloor 6-1\right) = 1$$

From Equation 28, and by calculating *T, a, b, c,* and *z* the threshold equals

$$ln\left(1+\frac{2}{23}\right)+\sum_{s=1}^{1} ln\left(1+\frac{5}{23+2+s*1+(s-1)5}\right)$$
$$+\,ln\left(1+\frac{1}{23+2+(1+1)1+(1*5)}\right)=$$
$$ln\left(1+\frac{2}{23}\right)+ln\left(1+\frac{5}{26}\right)+ln\left(1+\frac{1}{32}\right)\approx 0.290$$

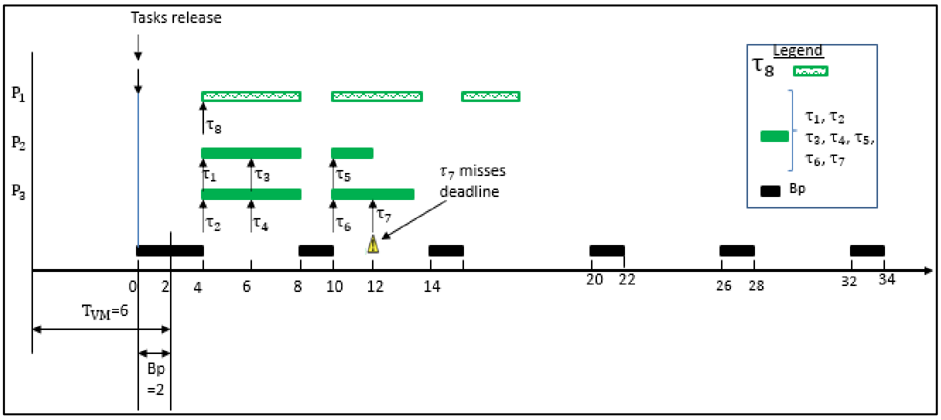


Figure 16 . The worst case scenario: Scheduling tasks with $m=3, T_{VM}=6, Bp=2$

Since the threshold is smaller than U'', i.e., $0.290<0.467$, we allocate τ_8 to its own processor and then check if it is schedulable. Since $C_8 \leq T_8-D_{VM}+T_{VM}-Bp-\left\lfloor\frac{T_8-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor Bp-min\left(T_8-D_{VM}+T_{VM}-Bp-T_{VM}\left\lfloor\frac{T_8-D_{VM}+T_{VM}-Bp}{T_{VM}}\right\rfloor, Bp\right)$, i.e., $10<26$, we see that τ_8 is schedulable. The next step is to check one by one if the remaining tasks $\tau_1, \tau_2, \dots \tau_7$ are schedulable on the remaining two processors using RM. It turns out that the task set is schedulable (see Figure 17).

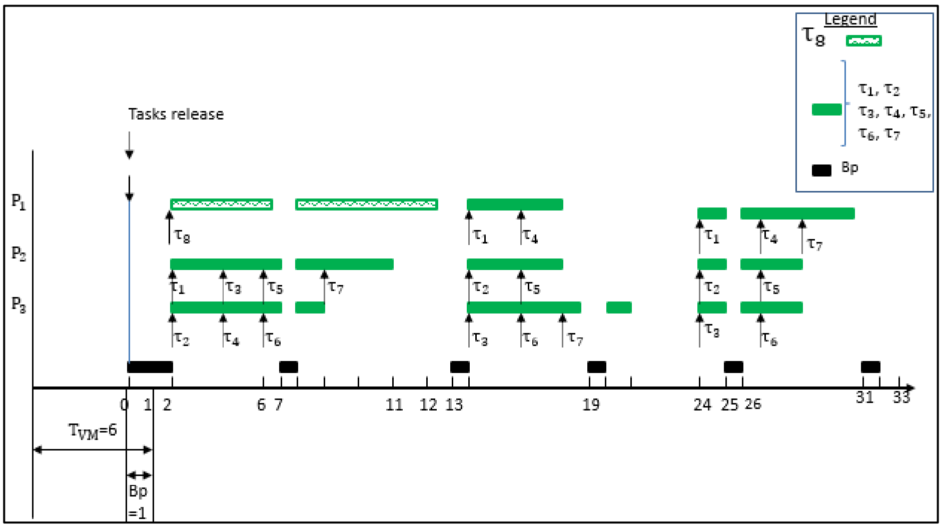


Figure 17. The worst case scenario: scheduling tasks with $m = 3, T_{VM} = 6, Bp = 1, D_{VM} = T_{VM}$

5.5.2 Hypervisor scheduling parameters trade-off

If we select a short D_{VM}, the jitter is reduced, which increases the chance that the task set Γ can be scheduled. However, decreasing D_{VM} will make it harder to schedule the VM on the hypervisor level in most cases (if the periods of the VMs sharing the hardware are all the same, or harmonic, increasing the D_{VM} may, however, not increase schedulability). In order to compensate for this increased scheduling difficulty on the hypervisor level one could either extend the T_{VM} or shorten the C_{VM} of the VM. However, extending the T_{VM} and/or shortening the C_{VM} will make it harder to schedule Γ on the VM. This means that there is a trade-off. One way of exploring this trade-off is to extend the D_{VM} and shorten the T_{VM} or vice versa, or to shorten both the D_{VM} and the C_{VM} or vice versa (still keeping $C_{VM} \leq D_{VM} \leq T_{VM}$). By using the results in this paper, these trade-offs can be evaluated for a certain Γ and for a certain set of VMs sharing the multiprocessor hardware (if a VM uses either all hardware resources or no hardware resources at each point in time, then scheduling of the VMs on the hypervisor level can be evaluated using well known schedulability tests from single-processors [6]).

There is of course some context switching overhead on the hypervisor level, i.e., there will be a context switching delay before the VM can start executing the real-time tasks. If we assume that the time for a context switch is fixed, it is trivial to incorporate the context switching overhead into our formulas since we keep track of the number of VM context switches in the worst-case scenarios. It is clear that a large context switching overhead will favor a long T_{VM} (since this will reduce the number of context switches). Based on the discussion above we know that a long T_{VM} can to some extent be compensated by a short D_{VM}. To sum up, the optimal combination of the scheduling parameters on the hypervisor level depends on the task set Γ, the context switching overhead, and whether the T_{VM} periods of the VMs sharing the hardware are harmonic or not; the effect of different combination of scheduling parameters on the hypervisor level can be evaluated using our results and well-known results from single-processor real-time scheduling.

5.6 Evaluation of Average Performance

As discussed before, the classification of tasks into heavy and light is the state-of-the-art way of avoiding Dhall's effect. By avoiding Dhall's effect we can guarantee that all task sets with a low utilization are schedulable. It is, however, not clear how the classification of tasks into heavy and light affects the average performance of a real-time system. In order to evaluate the average performance of using the heavy/light classification, we did an experiment.

We consider a multiprocessor VM with 10 processors, we work on two cases, (1) one task set has 300 tasks and (2) one task set has 500 tasks. Periods are randomly generated with a uniform distribution over a range of [100...3000]. Each case is simulated for different task set utilizations U [0.1, 0.2,…, 0.5] per processor. The distribution is done by the UUnifast algorithm, this algorithm efficiently generates task sets with uniform distribution with $O(n)$ complexity [26]. In order to avoid that individual tasks may have utilization higher than one, we generated n/m tasks m times (n is the number of tasks and m is the number of processors), and then we consolidated all tasks into a single task set with n tasks. A task's execution time was obtained by multiplying the utilization of the task with the task's period. We ran 30 task sets in each case and measured the average success ratio, which is the number of schedulable task sets over the total number of task sets.

Finding the optimal values for C_{VM}, D_{VM} and T_{VM} is complex. The optimal values do not only depend on the real-time task set being scheduled; they also depend on the system overhead for switching from one VM to another. Theoretically, it is always better to have very short T_{VM} [36]. The obvious problem with this is that for small T_{VM} we get excessive context switching between different VMs. This means that there is a trade-off where also the system overhead for context switching needs to be considered; for more details on this see [36]. The measurements are done for RM priority assignment and using heavy/light priority assignment. T_{VM} equals to a half of the shortest tasks' period in the task set in both cases; this has been heuristically found to be reasonable [36]. We have $Bp = C_{VM} = D_{VM} = T_{VM}/2$ in both cases.

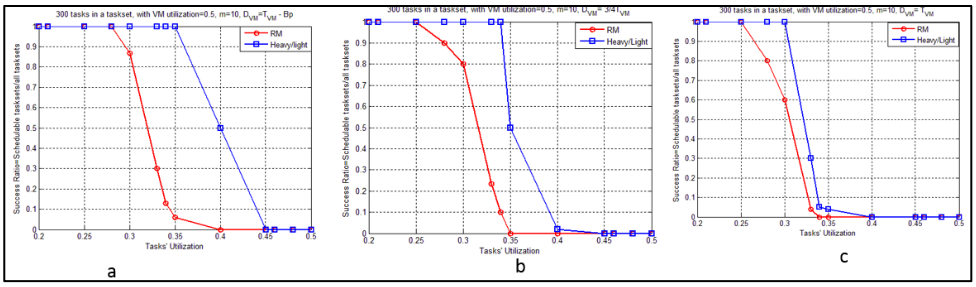


Figure 18. Case (1) with 300 tasks per task set, (a) $D_{VM} = T_{VM} - Bp$, (b) $D_{VM} = 0.75T_{VM}$, and (c) $D_{VM} = T_{VM}$

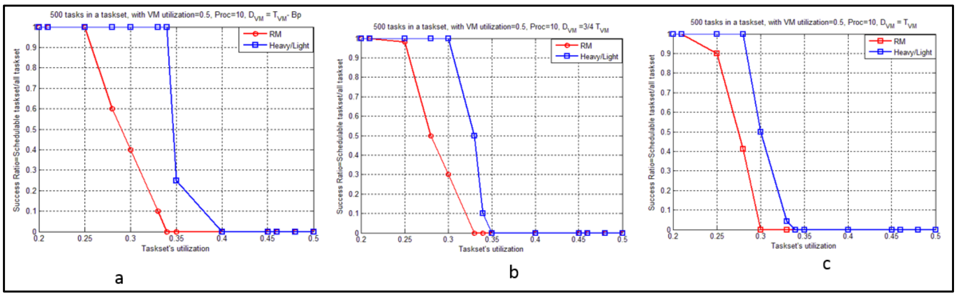


Figure 19. Case (2) with 500 tasks per task set, (a) $D_{VM} = T_{VM} - Bp$, (b) $D_{VM} = 0.75\ T_{VM}$, and (c) $D_{VM} = T_{VM}$

Figures 18 and 19 show that the utilization based schedulability test and heavy light priority assignment performs better than RM for both cases. When D_{VM} is as high as possible, i.e., $D_{VM} = T_{VM}$, the utilization based test schedules fewer task sets (see Figure 18 (c), 19 (c)). The best case is when D_{VM} is as short as possible, i.e., $D_{VM} = T_{VM} - Bp$ (see Figure 18 (a) and 19 (a)). However, when D_{VM} is short it will be more difficult to schedule the VMs so that all VMs meet their deadlines (as defined by D_{VM}). For instance, if we have two VMs with $D_{VM} = T_{VM} - Bp$, we will (unless for special cases with harmonic periods) not be able to schedule these two VMs on the target multiprocessor even if the sum of C_{VM}/T_{VM} is very small for these VMs. If we allow more jitter, it will be much easier to schedule these two VMs. This means that there is a trade-off; a short D_{VM} makes it easier to schedule real-time tasks inside the VM, but makes it harder to schedule the VM so that the VM deadlines are always met. Keeping the same utilization and increasing the number of tasks decreases the schedulability, i.e., the success ratio in Figure 18 (300 tasks) is higher than the corresponding success ratio in Figure 19 (500 tasks).

5.7 Conclusions

We present an algorithm that determines if a task set that executes in a VM with m virtual processors is schedulable or not. The test for each task in the task set is based on the utilization of the tasks with higher priorities, the period of the VM executing the task set (T_{VM}), the time that the VM is blocked in each period (Bp), and the VM deadline (D_{VM}). D_{VM} affects the length of the first period when the VM is blocked. In the worst case when $D_{VM} = T_{VM}$, the tasks start executing after two *Bps*. In the worst case when $D_{VM} = T_{VM} - Bp$, the tasks start executing after one *Bp*. Generally, we have $T_{VM} - Bp \leq D_{VM} \leq T_{VM}$ where tasks start executing after a time t, ($Bp \leq t \leq 2Bp$) depending on D_{VM}. A large D_{VM} makes it easier to schedule the VMs on the physical hardware, since the hypervisor has more flexibility regarding when the VM should be given access to the physical hardware. However, this flexibility makes the worst-case scenario more unfavorable for the real-time tasks running in the VM, so there is a trade-off. The preferred balance in this trade-off depends on the characteristics of the other VMs that share the same physical multiprocessor.

We calculate parameters *T, a, b, c*, and *z*, based on *m*, D_{VM}, T_{VM}, *Bp*, and the task τ_x (the task that we are testing). In the utilization-based schedulability test for τ_x, we compare the utilization U'' of the interfering tasks with the bound

$$U'' \leq ln\left(1+\frac{a}{T}\right)+\sum_{s=1}^{s=z} ln\left(1+\frac{b}{T+a+sBp+(s-1)b}\right) + ln\left(1+\frac{c}{T+a+(z+1)Bp+zb}\right)$$

If the utilization of the interfering tasks is above the bound, τ_x is classified as heavy and gets the highest priority, otherwise the task is classified as light. Light tasks are scheduled using RM priority assignment. Our algorithm checks each task in the task set and classifies each task as heavy or light. It is shown through simulation that the classification in light and heavy tasks used in the utilization-based schedulability test is more favorable than traditional RM priority assignment in the sense that we can on average schedule more task sets.
Our results makes it possible to evaluate important trade-offs and take informed decisions when selecting scheduling parameters on the hypervisor level, also considering the effect of context switching overhead at the hypervisor level. The computational complexity of the algorithm is $O(mn)$, where n is the number of tasks and m is the number of virtual cores (processors) in the VM running the real-time tasks.

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. We also thank Telenor Sverige for providing the data.

References

[1] Lundberg, L. and Shirinbab, S. (2013) Real-time scheduling in cloud-based virtualized software systems. Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies, Oslo, 01-03 September, pp. 54–58, ACM New York, NY, USA.

[2] Lundberg, L. (2002) Analyzing fixed-priority global multiprocessor scheduling. Proceedings of Eighth IEEE Real-Time and Embedded Technology

and Applications Symposium, San Jose, CA, USA, 25-27 September, pp. 145–153. IEEE Computer Society Washington, DC, USA.

[3] Buttazzo, G. C. (2011) Hard real-time computing systems: predictable scheduling algorithms and applications. Springer, NY, USA.

[4] Niyizamwiyitira, C., Lundberg, L. and Lennerstad, H. (2015) Utilization-based Schedulability Test of Real-time Systems on Virtual Multiprocessors. Proceedings of 44th International Conference on Parallel Processing Workshops, Beijing, 01-04 September, pp. 267–276. IEEE, New Jersey, USA

[5] Liu, C. L. and Layland, J. W. (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM, 20, pp. 46–61.

[6] Burns, A. and Wellings, A. J. (2001) Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX. Pearson Education, Boston, USA.

[7] Dhall, S. K. and Liu, C. L. (1979) On a real-time scheduling problem, Operational Research, 26, pp. 127–140.

[8] Abdelzaher, T., Andersson, B. and Jonsson, J. (2002) The aperiodic multiprocessor utilization bound for liquid tasks. Proceedings of Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 25-27 September, San Jose, CA, USA, pp. 173–184. Washington, DC, USA.

[9] Andersson, B., Baruah, S. and Jonsson, J. (2001) Static-priority scheduling on multiprocessors. Proceedings of Real-Time Systems Symposium, 25-27 September, San Jose, CA, USA, pp. 193–202. IEEE Computer Society Washington, DC, USA.

[10] Cucinotta, T., Anastasi, G. and Abeni, L. (2009) Respecting Temporal Constraints in Virtualised Services. Proceedings of COMPSAC, Seattle, WA, USA, 20-24 July, pp. 73–78. IEEE Computer Society Washington, DC, USA

[11] Zhu, H., Goddard, S. and Dwyer, M. B. (2011) Response Time Analysis of Hierarchical Scheduling: The Synchronized Deferrable Servers Approach. Proceedings of Real-Time Systems Symposium, Vienna, Austria, 29 Nov-2 Dec, pp. ˜239–248. IEEE Computer Society Washington, DC, USA.

[12] Shin, I., Easwaran, A. and Lee, I. (2008) Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors. Proceedings of Euromicro

Conference on Real-Time Systems, Prague, Czech Republic, pp. 181–190. IEEE Computer Society Washington, DC, USA.

[13] Lee, J. et al., (2012) Realizing compositional scheduling through virtualization, Proceedings of Real-Time and Embedded Technology and Applications Symposium, Beijing, China, pp. 13–22. IEEE Computer Society Washington, DC, USA.

[14] Lunniss, W., Altmeyer, S., Lipari, G. and Davis, R. I. (2014) Accounting for cache related pre-emption delays in hierarchical scheduling. Proceedings of the 22nd International Conference on Real-Time Networks and Systems, Versaille, France, 8-10 October, pp. 183-186. ACM New York, NY, USA.

[15] T. Cucinotta, T.,Giani, D., Faggioli, D., and Checconi, F. (2011) Providing performance guarantees to virtual machines using real-time scheduling. Proceedings of Euro-Par 2010 Parallel Processing Workshops, Ischia, Italy ,30 Aug, -3 Sept. pp. 657–664. Springer-Verlag Berlin Heidelberg.

[16] Chen, S., Phan, L. T., Lee, J., Lee, I., and Sokolsky, O. (2011) Removing abstraction overhead in the composition of hierarchical real-time systems. Proceedings of Real-Time and Embedded Technology and Applications Symposium, Chicago, IL, USA, pp. 81–90. IEEE Computer Society Washington, DC, USA.

[17] Phan, L. T., Xu, M., Lee, J., Lee, I., and Sokolsky, O. (2013) Overhead-aware compositional analysis of real-time systems. Proceedings of Real-Time and Embedded Technology and Applications Symposium, Philadelphia, PA, USA, pp. 237–246. IEEE Computer Society Washington, DC, USA.

[18] Xu, M. et al. (2013) Cache-aware compositional analysis of real-time multicore virtualization platforms. Proceedings of Real-Time Systems Symposium, Vancouver, BC, Canada, pp. 1–10. IEEE Computer Society Washington, DC, USA.

[19] Lundberg, L. (2002) Utilization based schedulability bounds for age constraint process sets in real-time systems. Real-Time Syst., 23, pp. 273–295.

[20] Lundberg, L. and Lennerstad, H. (2007) Guaranteeing response times for aperiodic tasks in global multiprocessor scheduling. Real-Time Syst., 35, pp. 135–151.

[21] Easwaran, A., Shin, I. and Lee, I. (2009) Optimal virtual cluster-based multiprocessor scheduling, Real-Time Syst., 43, pp. 25–59.

[22] Markatos, E. and LeBlanc, T. (1991) Predictable Virtual Processors for Real-Time Systems. Technical Report, Rochester Univ.

[23] Biondi, A., Buttazzo, G. and Bertogna, M. (2016) Partitioning and Interface Synthesis in Hierarchical Multiprocessor Real-Time Systems. Proceedings of the 24th International Conference on Real-Time Networks and Systems, Brest, France, pp. 257–266. ACM New York, NY, USA.

[24] Xi, S. et al. (2014) Real-time multi-core virtual machine scheduling in Xen. Proceedings of International Conference on Embedded Software, Jaypee Greens, India, pp. 1–10. IEEE Computer Society, Washington DC, USA.

[25] Lundberg, L. and Lennerstad, H. (2003) Global multiprocessor scheduling of aperiodic tasks using time-independent priorities, Proceedings of The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Ontario, Canada, pp. 170–180, Washington, DC, USA.

[26] Bini, E. and Buttazzo, G. C. (2005) Measuring the performance of schedulability tests, Real-Time Syst., 30, pp. 129–154.

[27] Burmyakov, A., Bini, E. and Tovar, E. (2012) The Generalized Multiprocessor Periodic Resource Interface Model for Hierarchical Multiprocessor Scheduling. Proceedings of the 20th International Conference on real-time and network systems (RTNS'12), pp. 131-139, Pont-à-Mousson, France.

[28] Bini, E., Bertogna, M. and Baruah, S. (2010) The Parallel Supply Function Abstraction for a Virtual Multiprocessor. Dagstuhl Seminar Proceedings 10071, http://drops.dagstuhl.de/opus/volltexte/2010/2542.

[29] Shin, I., Easwaran, A. and Lee, I. (2008) Hierarchical Scheduling Framework for Virtual Clustering Multiprocessors. Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp. 181-190, Prague, Czech Republic.

[30] Feng, X. and Mok, A. (2002) A Model of Hierarchical Real-Time Virtual resources, Proceedings of the 23rd IEEE Real-Time Systems Symposium, pp. 26-35, Austin, Texas, USA.

[31] Shin, I. and Lee, I. (2004) Compositional Real-Time Scheduling Framework, Proceedings of the 25th IEEE International Real-Time Systems Symposium, pp. 57-67, Lisbon, Portugal.

[32] Davis, R. and Burns A. (2009) Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems, Proceedings of the 30th IEEE Real-Time Systems Symposium, pp. 398-409, Washington, DC, USA.

[33] Kim, J., Gangadharan, D., Sokolsky, O., Legay, A. and Lee, I. (2017) Extensible Energy Planning Framework for Preemptive Tasks, Proceedings of the 20th International Symposium on Real-Time Computing, pp. 32-41, Toronto, Canada.

[34] Lundberg, L. (2011) Slack-based multiprocessor scheduling of aperiodic real-time tasks. Real-Time Systems 47(6), pp. 618-638.

[35] Andersson, B. and Jonsson, J. (2000) Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition, Proceedings of the 7th International Workshop on Real-Time Computing and Applications Symposium, pp. 337-346, Cheju Island, South Korea.

[36] Niyizamwiyitira, C. and Lundberg, L. (2017) Real-time scheduling of multiple virtual machines, International Journal of Computers and their Applications, Vol. 24, no. 3, pp. 91-109.