Achieving Real-Time in Distributed Computing:

From Grids to Clouds

Dimosthenis P. Kyriazis
National Technical University of Athens, Greece

Theodora A. Varvarigou
National Technical University of Athens, Greece

Kleopatra G. Konstanteli
National Technical University of Athens, Greece

Information Science
REFERENCE

Senior Editorial Director: Kristin Klinger
Director of Book Publications: Julia Mosemann
Editorial Director: Lindsay Johnston
Acquisitions Editor: Erika Carter
Development Editor: Joel Gamon
Production Editor: Sean Woznicki
Typesetters: Natalie Pronio, Jennifer Romanchak, Milan Vracarich, Jr.
Print Coordinator: Jamie Snavely
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Achieving real-time in distributed computing: from grids to clouds / Dimosthenis P. Kyriazis, Theordora A. Varvarigou, and Kleopatra Konstanteli, editors.
p. cm.
Includes bibliographical references and index.
Summary: “This book offers over 400 accounts from a wide range of specific research efforts on methodologies, tools, and architectures for complex distributed systems that address the practical issues of performance guarantees, timed execution, real-time management of resources, synchronized communication under various load conditions, satisfaction of QoS constraints, as well as dealing with the trade-offs between these aspects”-- Provided by publisher.
ISBN 978-1-60960-827-9 (hardcover) -- ISBN 978-1-60960-828-6 (ebook) -- ISBN 978-1-60960-829-3 (print & perpetual access) 1. Real-time data processing. 2. Electronic data processing--Distributed computing. I. Kyriazis, Dimosthenis P., 1973- II. Varvarigou, Theordora A., 1966- III. Konstanteli, Kleopatra, 1981-
QA76.54.A328 2011
004’.33--dc23
2011016061

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 11
Execution and Resource Management in QoS-Aware Virtualized Infrastructures

Dominik Lamp
University of Stuttgart, Germany

Sören Berger
University of Stuttgart, Germany

Manuel Stein
Alcatel-Lucent Bell Labs, Germany

Thomas Voith
Alcatel-Lucent Bell Labs, Germany

Tommaso Cucinotta
Scuola Superiore Sant'Anna, Italy

Marko Bertogna
Scuola Superiore Sant'Anna, Italy

ABSTRACT

Both real-time systems and virtualization have been important research topics for quite some time now. Having competing goals, research on the correlation of these topics has started only recently. This chapter overviews recent results in the research literature on virtualized large-scale systems and soft real-time systems. These concepts constitute the fundamental background over which the execution environment of any large-scale service-oriented real-time architecture for highly interactive, distributed, and virtualized applications will be built in the future. While many aspects covered in this chapter have already been adopted in commercial products, others are still under intensive investigation in research labs all over the world.

INTRODUCTION

Traditional real-time methodologies focus on hard real-time safety-critical systems, where applications have associated a set of temporal constraints (e.g., deadlines) which must never be violated, as otherwise the system as a whole will fail – with potentially fatal consequences such as loss of human lives. However, a class of soft real-time applications exists, for example in the multimedia domain, in which an approach to meet the desired timing constraints under all circumstances is neither needed nor practical. In fact, for those applications, violations of the temporal constraints lead to degrations in Quality of Service (QoS), rather than an entire system failure. Thus, they are

DOI: 10.4018/978-1-60960-827-9.ch011

usually tolerated as long as their frequency and severity remains within sensible limits.

Different service-oriented architecture (SOA) and cloud management approaches exist that possess specific features to provide increased service reliability, scalability, host virtualization, application failure detection, global resource management, and optimized server utilization. However, in case of deployment of real-time interactive distributed applications, such features need to be complemented by taking the timing constraints that are in place for the applications into proper consideration during the various decision processes and run-time mechanisms. The challenge that has to be addressed is that the goals of virtualization and real-time-compliance are competing: Virtualization is used to exploit multiplex gains by sharing physical resources among multiple tasks so that *statistically* the resource usage is maximized. On the other hand, real-time systems are designed to guarantee response times even for the worst case scenario, resulting in resource underutilization in the average case.

The remainder of this chapter is structured as follows. First, an overview of real-time scheduling is given. Section two discusses how virtualization mechanisms improve resiliency and utilization optimization. The third part of the chapter covers host virtualization mechanisms. The chapter concludes with an overview of mechanisms to give real-time guarantees in virtualized environments.

REAL-TIME SCHEDULING

Real-time theory and methodologies are gaining applicability in the field of soft real-time systems. In this domain, applications possess precise timing and performance requirements, but occasional failures in meeting them may be easily tolerated by the system, causing a graceful degradation in the quality of the provided service.

The real-time literature on soft real-time technologies for General purpose operating systems (GPOSes) is growing, and the major research branches are those in relation to: multiprocessor scheduling, soft real-time scheduling, QoS control in distributed real-time applications and adaptive QoS control, as detailed below.

Scheduling Real-Time Task Sets on Multiprocessor Platforms

Even if the concept of multiprocessing has always been present in the real-time community, only recently it is receiving a significant attention, thanks to the increasing industrial interest on such platforms, and their consequent increasing availability. While the scheduling problem for uniprocessor systems has been widely investigated for decades, producing a considerable variety of publications and applications, there are still many open problems regarding the schedulability analysis of multiprocessor systems. As pointed out by Liu in his seminal paper (C. L. Liu, 1969): "*few of the results obtained for a single processor generalize directly to the multiple processor case: bringing in additional processors adds a new dimension to the scheduling problem*".

Unfortunately, predicting the behaviour of a multiprocessor system requires in many cases a considerable computing effort. To simplify the analysis, it is often necessary to introduce pessimistic assumptions. This is particularly needed when modelling globally scheduled multiprocessor systems, in which the cost of migrating a task from one processor to another can significantly vary over time. The presence of caches and the frequency of memory accesses have a significant influence on the worst-case timely parameters that characterize the system. To bind the variability of these parameters, often real-time literature focuses on platforms with multiple processors but with no caches, or whose cache miss delays are known. Also, the cost of pre-emption and migration on multi-processor systems is a very important issue that still needs to be properly considered in real-time methodologies. Some research in the

domain of hardware architectures moves towards partially mitigating such issues. Recently, a few architectures have been proposed that limit penalties associated to migration and cache misses, for example the ARM's MPCore. Some researchers have recently proposed (Audsley & Bletsas, 2004; Furunäs, 2000; Kuacharoen, Shalan & Vincent J. Mooney III, 2003; Lindh, Stärner & Furunäs, 1995; Ward & Audsley, 2002) hardware implementations of some parts of the operating system, allowing one to reduce the scheduling penalties of multiprocessor platforms.

Approaches, Implementations and Comparisons

Scheduling on Multiprocessors

When deciding which kind of scheduler to adopt in a multiple processor system, there are two main options: *partitioned* scheduling and *global* scheduling:

Partitioned Scheduling

In a partitioned scheduler, there are multiple ready queues, one for each processor in the system, and it is possible to leave a processor in idle state even when there are ready tasks needing to execute. Each queue is managed according to (well-known) uni-processor scheduling algorithms, and task migration is not allowed. The placement of tasks among the available processors is a critical step. The problem of optimally dividing the workload among the various queues, so that the computing resources are well utilized, is analogous to the bin-packing problem, which is known to be NP-hard in the strong sense (Garey & Johnson, 1979; Leung & Whitehead, 1982). This complexity is typically avoided using sub-optimal solutions provided by polynomial and pseudo-polynomial time heuristics. Examples of policies used for this purpose are First Fit, Best Fit, Next Fit, and First Fit with decreasing utilizations (Burchard, Liebeherr, Oh & Son, 1995; Lauzac, Melhem & Mossé, 2003; López, Díaz & García, 2004; López & García, 2003).

Global Scheduling

For task sets with highly varying computational requirements, instead of wasting computational power by repeatedly invoking complex load-balancing algorithms, it is better to use a global scheduler (Andersson & Jonsson, 2000). This way, tasks are extracted from a single system-wide queue and scheduled onto the available processors. The load is thus intrinsically balanced, since no processor is idled as long as there is a ready task in the global queue.

A class of algorithms, called Pfair schedulers (Sanjoy Baruah, Neil Cohen, Greg Plaxton & Donald Varvel, 1996), is able to ensure that the full processing capacity can be used, but unfortunately at the cost of a potentially large run-time overhead. Migrative and non-migrative algorithms have been proposed (Anand Srinivasan & Sanjoy Baruah, 2002; Andersson & Eduardo Tovar, 2006; Baker, Cirinei & Bertogna, 2008; Bertogna, Michele Cirinei & Giuseppe Lipari, 2005) modifying well-known solutions adopted for the single processor case and extending them to deal with the various anomalies (Dhall & C. L. Liu, 1978) that arise on a parallel computing platform.

Complications in using a global scheduler mainly relate to the cost of inter-processor migration, and to the kernel overhead due to the necessary synchronization among the processors for the purpose of enforcing a global scheduling strategy. Even if there are mechanisms that can reduce the migration cost, such kind of schedulers could nevertheless cause a significant schedulability loss (i.e., the guaranteed bandwidth that is wasted due to pre-emption and migration overhead) when tasks have a large context associated. Therefore, the effectiveness of a global scheduler is rather conditioned by the application characteristics and by the architecture in use.

Hybrid Schedulers

In addition to the above classes, there are also intermediate solutions, like hybrid- and restricted-migration schedulers. A hybrid-migration scheduler (John M. Calandrino, James H. Anderson & Dan P. Baumberger, 2007) limits the number of processors among which a task can migrate in order to limit the number of caches in which the task image is present. In this way, a fewer amount of cache misses is expected, and the cost of migration and context changes is smaller. This method is more flexible than a rigid partitioning algorithm without migration, and it is particularly indicated for systems with a high number of processors (with tens of CPUs), where it would be very difficult and time-consuming to move a task image from a computing unit to a distant one.

A restricted-migration scheduler is instead a scheduler that allows task migration, but only at job boundaries, i.e., before a job starts executing or at the end of its execution. In this way, the amount of information that must be transferred while migrating a task from a processor to another is likely to be less than in the full migration case (Baruah & Carpenter, 2003). A similar method consists in using a global scheduler without pre-emption: in this way, once a job starts executing on a CPU, it is not possible to interrupt it until the end of execution, so that migration can never take place (Baruah, 2006). However, non-pre-emptive systems can have significant schedulability overhead, due to potential delays caused by long chunks of code that can execute without being interrupted.

Scheduling and Exclusive Access to Shared Resources

There are well-known efficient protocols to arbitrate the exclusive access to shared resources for real-time task sets scheduled on a single processor platform: Priority Inheritance (PI), Priority Ceiling Protocol (PCP), Dynamic Priority Ceiling Protocol (DPCP), and Stack resource Policy (SRP). Each one of these protocols allows solving the "priority-inversion" problem that takes place when a task is blocked by a lower priority one at the beginning of a critical section accessing a shared resource. The SRP ensures that a task can never be blocked once it started executing. This minimizes the number of context switches, allows all tasks to be executed on a single stack, and prevents deadlocks.

Soft Real-Time Scheduling

Different scheduling algorithms have been proposed to support the specific needs of soft real-time applications. A first important class approximates the Generalized Processor Sharing concept of a *fluid flow* allocation, in which each application using the resource marks a progress proportional to its weight. Among the algorithms of this class, we can cite Proportional Share (Ian Stoica et al., 1996) and Pfair (Sanjoy Baruah, Neil Cohen, Greg Plaxton & Donald Varvel, 1996). The underlying principles of a family of algorithms known as Resource Reservation schedulers (Luca Abeni & Giorgio Buttazzo; Raj Rajkumar, Kanaka Juvva, Anastasio Molano & Shuichi Oikawa, 1998) are similar. In the Resource Kernels project (Raj Rajkumar et al., 1998), the resource reservation approach has been successfully applied to different types of resources (including disk and network). Also, it is noteworthy to mention that the current scheduler in the main-stream Linux kernel, known as Completely Fair Scheduler (CFS), is basically a variation of the Proportional-Share idea. The Resource Reservation framework has been adapted to partitioned multiprocessor systems in (Sanjoy Baruah & Giuseppe Lipari, 2004) and (S. Baruah & G. Lipari, 2004) for respectively, the CBS and TBS servers.

Design of Distributed Real-Time Applications

The problem of designing scheduling parameters for distributed real-time applications has received a constant attention in the past few years. In (Neil C., 1995), the authors introduce a notion of transaction for real-time databases characterized by periodicity and end-to-end constraints and propose a methodology to identify periods and deadlines of intermediate tasks. A similar approach is taken in (Gerber, R., Seongsoo Hong & Saksena, M, 1995), in which activation periods of the intermediate tasks that comply with end-to-end real-time requirements are synthesized by an optimization problem. In (Dong-In Kang, Richard Gerber & Manas Saksena, 2000), the same idea is applied to soft real-time applications. In this case, the authors use performance analysis techniques to decide the bandwidth allocated to each task that attain a maximum latency and a minimum average throughput for the chain of computation.

Concerning the modelling of timing requirements of real-time applications, usually models similar to synchronous dataflow networks (Walid A. Najjar & Edward A., 1999) are used. As shown in (Shuvra S. Battacharyya & Edward A., 1996), synchronous data flow networks lend themselves to an effective code generation process, in which an offline schedule is synthesized that minimizes the code length and the buffer size. The models used in (L. Palopoli & T. Cucinotta, 2007) and (Steve Goddard & Kevin Jeffay, 2001) are also a special case of synchronous dataflow, but, due to the inherently distributed and dynamic nature of the considered applications, the aim is not an optimized offline scheduling of activities, but an efficient on-line scheduling mechanism.

Future Trends and Current Research Directions

Basically, scheduling solutions based on a partitioned scheduling approach seem the most promising, but traditional approaches like Fixed Priority or Earliest Deadline First exhibit usually a high number of pre-emptions per task, or an unacceptably short duration of the time-slice dedicated to each task execution in the schedule. This may be dramatically negative for the performance if the activity that is scheduled is a virtual machine that may experience further context changes on its own. Appropriate solutions in this field are probably based on an appropriate control of pre-emptability by the scheduler, so to achieve appropriate trade-offs between the overall performance (computational throughput) of the hosted activities and their responsiveness/latency when considered individually.

Resiliency, Recovery, and Utilization Optimization

Both resiliency and recovery target at achieving high availability rates. In Information and Communication Technology, availability is a metric for the probability of a system failure and is calculated from parameters like the (statistical) age-related failure rate of HW components for a system of given architecture (reliability structure) and the predicted time required to put a failed system back into operational mode. Therefore, the calculation uses stochastic methods for determining the overall probability of a given architecture of age-related failure rate of components. Mean Time Between Failures (MTBF) is the average time between failure of hardware components and is estimated by the respective manufacturer. The time it takes to repair or replace a defective component is called Mean Time To Repair (MTTR).

If MTBF and MTTR are known, the availability of a system can be estimated as

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

As a consequence, the availability depends on how fast a failing component can be replaced or repaired. A typical value for a carrier-grade system is 99.999%, often also referred to as "five nines availability".

When assessing the quality of an availability estimation, it is essential to also consider the time frame for which a certain maximum outage is guaranteed: Within a one year period, five nines would allow a maximum outage of about 5 minutes. That means that the prepared component (standby component) must be in operational mode within 5 minutes for a maximum of 1 failure per year. When five nines are required and the respective timeframe is a month, then the recovery time must be less than 30 seconds.

Both the desired service quality and fines for the case that the negotiated service quality is not achieved are usually specified in a Service Level Agreement (SLA). On the technical level, Quality of Service (QoS) mechanisms are implemented to ensure that the SLA is kept. Two important techniques to ensure QoS, namely redundancy and migration, are described in the following sections.

Approaches, Implementations and Comparisons

Redundancy

Redundant systems are built in a way that the failure of one or more components does not affect the overall system. This is done by replicating the system's critical components. The way that components are replicated and hold available can differ significantly. However, the key concepts are cold standby and hot standby.

Cold standby means that the prepared components are available, but are kept separate from the "live" infrastructure. Usually, cold standby components have to be manually integrated into the system in the event of failure.

As they are not actively participating in service delivery during normal operation, the failure of a cold standby component does not affect the overall service availability provided that the primary components are in service.

Hot standby on the other hand means that the standby component is an active part of the infrastructure of the system and that it is kept up and running, being able to step in at any time.

A hot standby system can be passive, which means that the backup components are not adding to the system's performance. As long as the active component is not failing, the standby component does not execute any designated task. However, the backup component might perform the same operations as the primary unit in order to be able to immediately take over at any point.

In active hot standby environments, all components are actually carrying out tasks. By using load balancing mechanism, the load is distributed among multiple components. In the case of failure, the tasks of the failed component are redistributed among the remaining components. If the system shall be able to run without degradation in such a scenario, then the system needs to be overprovisioned in the non-failure scenario.

A variation of hot standby is used in critical systems: Here, multiple active systems perform the same task in parallel. The result of those systems is compared and considered valid only if the majority of systems provide the same result.

Depending on the required resiliency level, redundancy can be achieved in different ways. For critical components that are crucial for the overall availability, protection by dedicated standby components (1+1, 1:1 redundancy) is common. For less critical components, N active components of the system can be protected by a single standby component (N+1, N:1 redundancy).

In both cases, after recovery from the failure the original component might take over processing again (1:1/N:1) or might become a standby component (1+1/N+1).

The advantage of 1+1/N+1 redundancy is that only one disruption occurs. In 1:1/N:1 configurations, two disruptions take place; one when the fault occurs and a second after recovery when

a fall forward to the original system occurs. If the original system suffers from an intermittent failure, fall back and fall forward also occur intermittently, thus potentially disrupting the service multiple times.

Frequently, the term 1+1 is also used to refer to an active/active setup, since in most cases these two cases go hand in hand.

Basically, the same also applies to N:M and N+M, respectively. Here, N active components/links are protected by M redundant components/links. The difference is that in the "N+M" case, any system can replace any other. After repair of the failed one, there is no need to return back. The repaired system will become standby for all the active ones. In the "N:M" case, any system can replace any of the components, but only once. After repair of the failing system, a reversion is needed to get back to the original redundancy level.

In addition to the different means to provide additional resources to provide the desired QoS even in the case of failure, it also needs to be discussed how the take over process is prepared. This preparation process is heavily application-dependant, but three main concepts exist:

- *Stateless replacement.* The replacing system does not care about any states of the failing system. It is just brought in operation mode. This is normally the fastest way to come back to active operation mode. Existing sessions might be lost and need to be re-established. Thus, stateless replacement is best suited for stateless processes and short-running processes that can be easily re-run. The MTTR phase ends when the standby system has become active. Examples include web-servers serving static information.
- *Stable state protection.* During normal operation mode of active system, the stable states are synchronized to a standby system or to an external database. When the failover occurs to standby, the standby system will become active immediately, initialized with the stable state information or, after the states are recovered from this external database. The unstable states are lost. As an example for communication protocol stacks in telecom environment, this is known as stable call preservation. It is e.g. required that after a switchover 90% of the calls are preserved and just 10% of the transient calls are lost. The replacement scenario phase ends when standby system is able to run with the 90% of stable calls and accepting new calls.
- *Seamless protection.* normally goes hand in hand with an active/active operational mode, but it covers a different scope. The protected system is continuously exchanging the state changes to the standby system, so that, in case of a failing, the standby can immediately overtake the task from the failing one without losing any states. If there is any outage time during switchover, the replacement scenario phase ends when the standby is operational for the transferred tasks.

Migration

Migration is the relocation of running computational processes from one resource to another. This migration can be motivated by environmental changes that make it beneficial (cheaper, improve overall performance, etc.) to reallocate a certain process or bundle of processes. Migration however will impose an overhead either in time, resources utilization or costs that needs to be taken into account.

There are two different types of migration, *Single Process Migration* and *Virtual Machine Migration*.

Single Process Migration means that a single process is moved across different nodes. In order to migrate at process level, a system is needed that is able to reallocate processes among machines in a transparent way. One of the first examples is

the Sprite implementation (Douglis & Ousterhout, 1991) for SPARCstation 1 workstations. The proposed mechanism is used to offload computationally expensive tasks to idle workstations within the same domain. This is done by spawning processes on remote workstations and by migrating them to idle machines as soon as the local user requires the machine's resources.

Virtual Machine Migration: The use of virtualization enables the migration of processes by replicating the whole environment, including the operating system, they run in. Although certain computing overhead is inherent to this approach, a lot more flexibility is achieved, as no specific requirements are imposed to the running code or to the host environment (except for the support of virtualization). While this approach has gained quite some popularity with the open source Xen hypervisor (Ian Pratt, 2003), *live migration* is nowadays available in most modern virtualization solutions for x86 platforms. Current implementations do not support migration beyond the borders of the ISO/OSI Layer 2 Network the virtual machine is connected to.

Migration is also a suitable technology to avoid some outage scenarios. As an application running in an environment that allows migration is not bonded to a dedicated physical host, it can be moved to another location when an outage of the hardware is imminent. Thus, a physical host can be freed of all tasks before it is taken down for maintenance or before a critical condition (e.g., a faulty fan or a hard drive showing recoverable errors) becomes terminal. On a large scale, migration technology can even be used to evacuate complete data centres either to save energy by concentrating computational power in one location and shut down completely other locations or in the case of predicted disasters that might affect a whole area.

To support migration of processes or operating system instances, it is common to utilize virtualization technology as introduced in the upcoming section.

Virtualization

Virtualization is a key building block of modern IT infrastructures as it increases system resiliency, eases error and disaster recovery, and is a key pillar of modern systems to optimize system utilization. It is used to hide actual physical resources and to provide logical resources instead. This enables the use of one single physical resource to be rendered as several of them (1:n), the use of several physical resources to be rendered as a single logical resource (n:1) or the use of several physical resources to render several logical resources (n:n). Decoupling the fixed physical resources from the resources that are used in first instance enables great flexibility in the dimensioning of the resources needed. Resources can be created on demand tailored for a specific use.

A key aspect of virtualization is that the kind of resources is not changed, i.e., the virtual resources are equivalent to the underlying physical resources. If resources which differ from the available physical resources are required, a different technology called *emulation* is used. An emulator fully resembles a resource. Common examples are the emulation of cellular handsets on a desktop PC, the emulation of end-of-life hardware, or of hardware like graphics cards or network adapters for which operating systems provide driver support out of the box.

Approaches, Implementations and Comparisons

Platform virtualization enables the virtualization of a whole Operating System environment. In this case, one can distinguish the Host OS on top of which the virtualized Guest OSs run. Regarding virtualization for software resources, the following distinctions can be made:

- *Application virtualization:* Virtualizes resources for certain applications, with a vir-

tualization layer between the OS and the application.

- *OS-level virtualization:* All resources are virtualized at the OS level, i.e., the OS running on the physical hardware "duplicates" itself and provides multiple instances of itself. *OS-level virtualization* can be restricted to the OS kernel or also include system libraries and tools.
- *Software-based virtualization:* The operating system is run under the control of a virtualization program. While operations that cannot affect other processes are executed 1:1 on the underlying hardware, operations that could have side effects are intercepted and transformed into a safe command sequence.
- *Hardware-assisted virtualization:* Similar to the *Native Virtualization* scenario, the *Hardware* layer provides special support to for virtualization through a concrete set of instructions, e.g. Intel VT-d, AMD IOMMU. Usually, some hardware like network cards is emulated or paravirtualized.
- *Paravirtualization:* Does not fully simulate the hardware but an API is provided to the guest OS to access the physical resources. To this end, the guest system needs to be modified.

Often multiple of the above mechanisms are combined: While the core components of a system (like CPU and memory) are virtualized, other components like network adapters might be emulated or paravirtualized. It is common to restrict paravirtualization to I/O-intensive resources as paravirtualization provides the highest performance gain for these devices.

Some examples are:

- *Java* (Gosling & McGilton, 1996) is a programming language developed at Sun Microsystems. It fundamental difference from programming popular at the time of its creation is the fact that its compiler does not produce binaries native to a certain processor, but so-called byte-code that is interpreted by the Java Virtual Machine at runtime. As a result, Java programs are platform independent and can be run on any host platform that provides the appropriate runtime environment.
- *.NET* is a framework developed by Microsoft (Microsoft Corporation). It is very similar to Java in that source code is compiled to an intermediate form, the common language runtime, that is executed by a runtime environment independent of the underlying system. Programs for .NET can be written in a number of languages such as C#, C++, or Visual Basic.
- *Solaris Zones* are an integral concept of the Solaris operating system. Zones basically are execution environments within a single instance of Solaris OS (Menno Lageman, 2005). While Zones share the same (running) Kernel, they are isolated against each other.
- *FreeBSD Jails* have been introduced in (Kamp & Robert N. M. Watson, 2000). The jail(2) system call of the FreeBSD kernel can be used to jail processes into separate environments. It limits access to a certain area of the file system as well as to other resources such as IP addresses.
- *Virtuozzo* (Parallels Holdings Ltd.) is a product that separates an operating system into several logical containers. It is thus similar to the Zones and Jails concepts described above. Virtuozzo is available for Linux and Windows. Parts of the Linux implementation are also available as open source under the name *OpenVZ*.
- *VMWare,* a software family developed by VMWare, Inc. The company's first product, VMWare Workstation, was released in 1999 and actually the first solution to bring virtualization to the x86 platform. Over

time, the product range has extended to hypervisor-based products for server virtualization (VMWare Inc.). Current releases support live migration of virtual machines as well as fault tolerance through replicated, parallel execution of virtual machines on multiple physical hosts (VMware, 2009).

- *VirtualBox* is a cross-platform virtualization software that runs on top of Windows, Linux, Mac OS, and OpenSolaris operating systems. It is distributed as both Open Source under the GPL and with an extended feature set under a proprietary license (Oracle Corporation).
- *Kernel-based Virtual Machine (KVM),* which is another full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, which provides the core virtualization infrastructure, and a processor specific module. KVM also requires a modified QEMU although work is underway to get the required changes upstream. Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc. KVM is a free open-source software (Qumranet).
- *Xen,* which is an x86 virtual machine monitor which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion but without sacrificing either performance or functionality. Although XenSource Inc. sells enterprise versions of the software, the company also support open source project. The research originated at the University of Cambridge led by Ian Pratt (Ian Pratt, 2003).

Virtualized Real-Time Service-Oriented Infrastructures

When Service-Oriented Infrastructures are built using virtualized environments, the temporal interferences among multiple virtual machines concurrently running on the same physical node may completely disrupt the quality of service of the hosted applications. Therefore, appropriate node-level mechanisms are needed in order to guarantee correct scheduling of concurrent virtualized real-time services within the same node. The need for real-time support within SOAs is witnessed by the RTSOA paradigm (Cucinotta, Anastasi & Abeni, 2008; McGregor & Eklund, 2008), and by the increasing need for real-time support within the Grid community (Cuzzocrea, 2008). Unfortunately, most of the works in these directions do not consider time-shared or virtualised nodes. For example, (P.A. Dinda et al., 2008) proposed the use of time-shared systems, but their work did not address the issues concerned with low-level real-time scheduling algorithms. Steps in this direction have been moved by (L. Almeida et al., 2008), who applied real-time scheduling theory to the problem of guaranteeing temporal guarantees to distributed applications built as a network of composeable services, focusing on the distribution aspects. Also, (Cucinotta et al., 2009) recently proposed a real-time service-oriented architecture for distributed real-time applications in the industrial automation context.

The problem of QoS provisioning to virtualized applications has been addressed in some previous work, but the level of determinism needed to run real-time applications inside a virtual machine has not been reached yet.

For example, Xen (Ian Pratt, 2003) uses an EDF-based reservation mechanism (called S-EDF) to enforce temporal isolation between the different VMUs. However, the S-EDF scheduler lacks a solid theoretical foundation, and is not guaranteed to work correctly in presence of dynamic activations and deactivations. As a result, it

seems to have problems in controlling the amount of CPU allocated to the various domains. In fact, in (Freeman, Foster I. T. et al., 2006), it is shown that the Xen scheduler is not able to properly control CPU allocations for I/O intensive operations.

If virtual machines are scheduled using proper real-time algorithms, the entire ensemble constituted by the host and the guests Operating Systems can be modelled as a hierarchy of schedulers. Therefore, its real-time performance can be evaluated by using hierarchical scheduling analysis techniques, such as the one proposed by Saewong and Rajkumar who extended resource reservations to support hierarchical reservations (Saewong, Rajkumar, Lehoczky, Klein & M. H., 2002). Shin and Lee proposed a different approach based on a compositional real-time scheduling framework (Shin & Lee, 2004), where the timing requirements of complex real-time components are analysed in isolation and subsumed into an abstract specification called interface, then combined to check schedulability of the overall system. A multiprocessor extension of this model has been presented in (Bini, Bertogna & Baruah, 2009; Bini, Buttazo & Bertogna, 2009).

Mok and others (Feng & Mok, 2002; Mok & Feng, 2001) presented a general methodology for hierarchical partitioning of a computational resource, where schedulers may be composed at arbitrary nesting levels. Specifically, they associate to each resource partition a characteristic function that identifies, for each time window of a given duration, the minimum time that the processor is allocated to the partition. On the other hand, (Lipari & Bini, 2004) addressed the problem of how to optimally tune the scheduling parameters for a partition, in order to fulfil the demand of contained real-time task sets. The latter technique has been applied by (Cucinotta et al., 2008) to analyse the schedulability of real-time tasks running in a virtualized Operating System, and to compute suitable scheduling parameters at the root scheduling level, complementing experimental results which showed how virtualized services may actually be run with predictable QoS levels on Linux by using a soft resource-reservation scheduler built inside the kernel. The more complex problem of considering applications that may share global resources has been considered by (Bertogna, Fisher & Baruah, 2009).

Future Trends in QoS-Compliant Virtualization

There is a lot of effort in the direction of allowing more and more different environments to be able to virtualize a greater number of virtualized guests with maximum performance. Another trend is the development of means to enable the hot-dynamic adjustments of virtualized resources at run time.

The virtualization trend opens a new bunch of possibilities in the field of customer-tailored services tied to an on-the-fly purpose specific SLA, specially the real-time component comes on stage. To this end necessary mechanisms need to be created to allow the use of resources with real-time constrains, independently of whether these resources are network resources or software.

Resource Optimization

In traditional real-time literature, the criticality of tasks and the consequent hard nature of their deadlines leave the problem of resource usage or optimization only as a secondary concern. However, when focusing on soft real-time systems, the research efforts shift completely on the problem of finding appropriate trade-offs between an efficient/optimum resource usage and the guarantees provided to the real-time activities. In this context, QoS optimization and adaptive QoS control for soft real-time tasks are particularly relevant research areas, as detailed below.

Approaches, Implementations and Comparisons

Global QoS Control

Concerning the choice of optimum scheduling parameters for maximizing the QoS of a distributed system where multiple applications may share different resources, (Ragunathan Rajkumar, Chen Lee, John P. Lehoczky & Daniel P. Siewiorek) proposed Q-RAM, a very general model that allows each task to be associated with a custom utility function which relates the resource consumption to the QoS experienced by the application. An optimization algorithm is then run to decide a recommended vector of resource allocations that optimizes the overall utility function of the system. In the original paper this is a static allocation scheme, for it is very computationally demanding (the optimization problem is NP-Hard). In (Sourav Ghosh, Jeffery Hansen, Ragunathan Rajkumar & John Lehoczky), the authors adapt the Q-RAM approach to a radar system. Another interesting work is described in (Fumiko Harada, Toshimitsu Ushio & Yukikazu Nakamoto, 2007), where an on-line adaptation algorithm based on a discrete integral controller, executed at periodic sampling instants, decides the resource allocation based on the knowledge of upper and lower bounds to the utility function. The approach in (L. Palopoli & T. Cucinotta, 2007) considers instead a more specific class of applications, for which the QoS is tightly related to the latencies and to the delays (thus no utility function is needed), trading generality for accuracy in QoS control, and it uses very efficient control algorithms that need to be activated potentially for each job of the application.

Adaptive QoS Control

The motivation for research on feedback-based scheduling is that a static design can be highly inefficient when applications exhibit high time-varying workloads. For example, for multimedia applications, the extensive use of compression technologies leads to a high variability in the computational as well as network/disk workloads, despite the typical periodic nature of the application.

In order to keep under control the evolution of application QoS parameters, it is possible to use two main approaches: application-level and resource-level adaptation. In the first case (Hideyuki Tokuda & Takuro Kitayama, 1993; Scott Brandt & Gary Nutt, 2002; Tatsuo Nakajima, 1998; Wust, Steffens, Bril & Verhaegh, 2004), in response to time-varying application requirements and availability of shared resources, a software infrastructure (typically a middleware) influences run-time parameters of the applications to make their requirements fit in the instantaneous resource availability.

Many papers in the past few years propose to govern the resource allocation based on a constant monitoring of the QoS experienced by the application. The application, in principle, can run almost unaware of the underlying ongoing adaptation. In (Anton Cervin, Johan Eker, Bo Bernhardsson & Karl-Erik, 2002) the authors propose to adjust the scheduling priorities to maximize the performance of a set of feedback controllers. Recently, in (Fumiko Harada et al., 2007; Sourav Ghosh et al.) a work inspired to the Q-RAM framework has been done. The idea is to associate each task with a custom utility function relating the resource consumption to the QoS. An on-line optimization problem is then solved to compute the optimal assignment of resources.

Focusing on soft real-time applications, in (Lu, Stankovic, Tao & Son, 2002) the authors use an EDF scheduling algorithm trying to dynamically adapt the deadline miss ratio. A research line based on a well-founded dynamical model of the system, from a QoS evolution perspective, was first introduced in (Luca Abeni, Luigi Palopoli, Giuseppe Lipari & Jonathan Walpole, 2002) for a single resource. The model introduces a QoS metric based on measurements of the finishing time of the jobs, and relies on an underlying resource-reservation scheduling policy. The authors propose

to split the controller into a prediction component and a feedback correction action, while the work is extended to a stochastic characterization of applications in (Cucinotta T., Palopoli L. & Marzario L., 2004), and to distributed applications in (L. Palopoli & T. Cucinotta, 2007).

A similar approach was the one proposed in (Ashvin Goel, Jonathan Walpole & Molly Shor), where a controller is used to regulate the progress rate of each task (real-rate scheduling), defined as the difference between a time-stamp associated to a computation and the actual time this computation is performed at. The approach is generalized to pipelines of tasks in (Steere, Shor, Goel & Walpole, 2000).

Finally, many papers have considered the problem of QoS adaptation from a software architecture point of view. For instance, the use of adaptation techniques inside general-purpose Operating Systems is discussed in (John Regehr & John A. Stankovic). In other papers, some authors propose to perform resource adaptation in a middleware layer (Eric Eide, Tim Stack, John Regehr & Jay Lepreau, 2004; Gill et al., 2005; Shankaran, Koutsoukos, Schmidt, Xue & Lu, 2006a). The latter work revolves around the QuO (Krishnamurthy et al., 2001) middleware framework, which is particularly noteworthy for it utilizes the capabilities of CORBA to reduce the impact of QoS management on the application code. A CORBA-oriented approach may also be found in (Shankaran, Koutsoukos, Schmidt, Xue & Lu, 2006b), where traditional control theory based on linearization, proportional control and linear systems stability is applied to the context of controlling resources allocation for a real-time object tracking system.

Future Trends

A good design for a soft real-time system is based on appropriate trade-offs between interactivity and responsiveness of individual applications or software components, and global utilization of the available resources. Basically, this leads to an overbooking of resources, usually based on statistical analysis of the application behaviour. However, whenever SLA agreements come in the loop, it may happen that a deadline miss, or the impossibility to sustain a certain throughput due to temporary or persistent overload conditions, even if not fatal for the system, may involve monetary consequences due to the presence of SLA agreements between service providers and end users.

The possibility not only to dynamically fine-tune the resources allocation for a given activity on a given host, but also to dynamically migrate activities on more powerful hosts opens up a new way of adapting resource availability to application requirements, based on the optimization of cost functions that may mix-in technical ingredients (expected capability for applications to meet their timing constraints) as well as business related ones (cost of occupying extra-resources out of the original plan, expected losses due to deadline misses).

CONCLUSION

Both Real-Time systems and virtualized infrastructures are already fairly well understood by themselves. The combination of these two paradigms, however, still poses quite some challenges. As most services do not require hard Real-Time constraints but have "only" soft Real-Time constraints, Real-Time aware service-oriented infrastructures that provide services on top of virtualized infrastructure are likely to become widely available in the not too distant future. This will enable also smaller companies to satisfy peak loads by exploiting the cost-effectiveness of using cloud resources while still being in control of the customer experience.

REFERENCES

Abeni, L., & Buttazzo, G. (1998). Integrating multimedia applications in hard real-time systems. *Proceedings of the IEEE Real-Time Systems Symposium,* December 1998, Madrid.

Abeni, L., Palopoli, L., Lipari, G., & Walpole, J. (2002). Analysis of a reservation-based feedback scheduler. *Proc. of the Real-Time Systems Symposium*.

Almeida, L., et al. (2008). Solutions for supporting composition of service-based RT applications. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing* (pp. 42–49).

Andersson, B., & Jonsson, J. (2000). *Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition* (pp. 337–346). Cheju Island, South Korea: RTCSA.

Andersson, B., & Tovar, E. (2006). Multiprocessor scheduling with few preemptions. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications (RTCSA)*.

Audsley, N., & Bletsas, K. (2004). Fixed priority timing analysis of real-time systems with limited parallelism. In *Proceedings of the Euromicro Conference on Real Time Systems*. Catania, Italy.

Audsley, N. C., Burns, A., Richardson, M. F., & Wellings, A. J. (1995). Data consistency in hard real-time systems. *Informatica, 19*(2).

Baker, T. P., Cirinei, M., & Bertogna, M. (2008). EDZL scheduling analysis. *Real-Time Systems: The International Journal of Time-Critical Computing*, *40*(3), 264–289.

Baruah, S. (2006). The non-pre-emptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems: The International Journal of Time-Critical Computing*, *32*(1-2), 9–20.

Baruah, S., & Carpenter, J. (2003). Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of the EuroMicro Conference on Real-time Systems*. Porto, Portugal: IEEE Computer Society Press.

Baruah, S., Cohen, N., Plaxton, G., & Varvel, D. (1996). Proportionate *progress: A notion of fairness in resource allocation*.

Baruah, S., & Lipari, G. (2004). *A multiprocessor implementation of the total bandwidth server*. International Parallel and Distributed Processing Symposium (IPPDS 04), Santa Fe.

Baruah, S., & Lipari, G. (2004). *Executing aperiodic jobs in a multiprocessor constant-bandwidth server implementation*. Euromicro Conference on Real-Time Systems (ECRTS 04), Catania (Italy).

Battacharyya, S. S., Lee, E. A., & Murthy, P. K. (1996). *Software synthesis from dataflow graphs*. Kluwer Academic Publishers.

Bertogna, M., Cirinei, M., & Lipari, G. (2005). New schedulability tests for real-time tasks sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*. Pisa, Italy: IEEE Computer Society Press.

Bertogna, M., Fisher, N., & Baruah, S. (2009). Resource-sharing servers for open environments. *IEEE Transactions on Industrial Informatics*, *5*(3), 202–220. doi:10.1109/TII.2009.2026051

Bini, E., Bertogna, M., & Baruah, S. (2009). Virtual multiprocessor platforms: Specification and use. In *Proceedings of 30th IEEE Real-Time Systems Symposium*.

Bini, E., Buttazo, G., & Bertogna, M. (2009). The multi supply function resource abstraction for multiprocessors: The global EDF case. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*.

Brandt, S., & Nutt, G. (2002). *Flexible soft real-time processing in middleware*. Real-Time Systems Journal, Special Issue on Flexible Scheduling in Real-Time Systems.

Burchard, A., Liebeherr, J., Oh, Y., & Son, S. H. (1995). New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, *44*(12), 1429–1442. doi:10.1109/12.477248

Calandrino, J. M., Anderson, J. H., & Baumberger, D. P. (2007). A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of the Euromicro Conference on Real-Time Systems.* Pisa.

Cervin, A., Eker, J., Bernhardsson, B., & Arzen, K.-E. (2002). Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, *23*(1/2). doi:10.1023/A:1015394302429

Cucinotta, T., Anastasi, G., & Abeni, L. (December 2008). Real-time virtual machines. In *Proceedings of the 29th Real-Time System Symposium (RTSS2008)*.

Cucinotta, T., Mancina, A., Anastasi, G., Lipari, G., Mangeruca, L., Checcozzo, R., & Rusinà, F. (2009). A real-time service-oriented architecture for industrial automation. *IEEE Transactions on Industrial Informatics, 5*(3).

Cucinotta, T., Palopoli, L., & Marzario, L. (2004). Stochastic feedback-based control of QoS in soft real-time systems. In *Proceedings of the 43rd IEEE Conference on Decision and Control*.

Cuzzocrea, A. (2008). Towards RT data transformation services over grids. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference* (pp. 1143–1149).

Dhall, S. K., & Liu, C. L. (1978). On a real-time scheduling problem. *Operations Research*, *26*, 127–140. doi:10.1287/opre.26.1.127

Dinda, P. A. (2008). Resource virtualization renaissance. *Computer*, *38*(5), 28–31.

Douglis, F., & Ousterhout, J. (1991). Transparent process, igration: Design alternatives and the Sprite implementation. *Software, Practice & Experience*, *21*(8), 757–785. doi:10.1002/spe.4380210802

Eide, E., Stack, T., Regehr, J., & Lepreau, J. (2004). Dynamic CPU management for real-time middleware-based systems. *Proceedings of 10th IEEE Real-Time and Embedded Technology and Applications Symposium*.

Feng, X., & Mok, A. K. (2002). A model of hierarchical real-time virtual resources. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*.

Freeman, T., & Foster, I. T. (2006). *Division of labor: Tools for growing and scaling grids* (pp. 40–51). ICSOC.

Furunäs, J. (2000). Benchmarking of a real-time system that utilises a booster. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications,* PDPTA2000.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York, NY: W. H. Freeman and Company.

Gerber, R., Hong, S., & Saksena, M. (1995). Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, *21*(7). doi:10.1109/32.392979

Ghosh, S., Hansen, J., Rajkumar, R., & Lehoczky, J. (2008). Integrated resource management and scheduling with multi-resource constraints. *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS04)*.

Gill, C. D., Gossett, J. M., Corman, D., Loyall, J. P., Schantz, R. E., Atighetchi, M., & Schmidt, D. C. (2005). *Integrated adaptive (QoS) management in middleware: A case study, real-time systems.*

Goddard, S., & Jeffay, K. (2001). Managing latency and buffer requirements in processing graph chains. *The Computer Journal*, *44*(6). doi:10.1093/comjnl/44.6.486

Goel, A., Walpole, J., & Shor, M. (2004). Real-rate scheduling. *Proceedings of Real-time and Embedded Technology and Applications Symposium*.

Gosling, J., & McGilton, H. (1996). *The Java™ language environment: A White Paper*. Mountain View.

Harada, F., Ushio, T., & Nakamoto, Y. (2007). Adaptive resource allocation control for fair QoS management. *IEEE Transactions on Computers*, *56*(3). doi:10.1109/TC.2007.39

Kamp, P.-H., & Watson, R. N. M. (2000). Jails: Confining the omnipotent root. In *Proc. 2nd Intl. SANE Conference*.

Kang, D.-I., Gerber, R., & Saksena, M. (2000). Parametric design synthesis of distributed embedded systems. *IEEE Transactions on Computers*, *49*(11).

Krishnamurthy, Y., Kachroo, V., Karr, D. A., Rodrigues, C., Loyall, J. P., Schantz, R. E., & Schmidt, D. C. (2001). *Integration of QoS-enabled distributed object computing middleware for developing next-generation distributed application*. LCTES/OM.

Kuacharoen, P., Shalan, M. A., & Mooney, V. J., III. (2003). A configurable hardware scheduler for real-time systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*.

Lageman, M. (2005). *Solaris containers: What they are and how to use them* (Sun blueprints online). Retrieved from http://www.sun.com/blueprints

Lauzac, S., Melhem, R., & Mossé, D. (2003). An improved rate-Monotonic admission control and its application. *IEEE Transactions on Computers*, *58*(3).

Leung, J. Y. T., & Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, *2*, 237–250. doi:10.1016/0166-5316(82)90024-4

Lindh, L., Stärner, J., & Furunäs, J. (1995). From single to multiprocessor real-time kernels in hardware. In *Proceedings of the IEEE Real Time Technology and Applications Symposium.* Chicago.

Lipari, G., & Bini, E. (2004). A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, *1*(2).

Liu, C. L. (1969). Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary*, *37*(60), 28–31.

López, J. M., Díaz, J. L., & García, D. F. (2004). Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, *28*(1), 39–68.

López, J. M., & Garcia, M. (2003). Utilization bounds for multiprocessor rate-Monotonic scheduling. *Real-Time Systems: The International Journal of Time-Critical Computing*, *24*(1), 5–28.

Lu, C., Stankovic, J., Tao, G., & Son, S. (2002). Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal on Control-Theoretic Approaches to Real-Time Computing, 9*.

McGregor, C., & Eklund, J. M. (2008). RT SOAs to support remote critical care: Trends and challenges. In *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference* (pp. 1199–1204).

Microsoft Corporation. (2010). *NET framework conceptual overview. NET framework 4*. Retrieved from http://msdn.microsoft.com/ library/ zw4w595w.aspx

Mok, A. K., & Feng, X. A. (2001). Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*.

Najjar, W. A., Lee, E. A., & Gao, G. R. (1999). Advances in the dataflow computational model. *Parallel Computing*, *25*, 13–14. doi:10.1016/ S0167-8191(99)00070-8

Nakajima, T. (1998). *Resource reservation for adaptive QoS mapping in real-time mach.*

Oracle Corporation. (n.d.). *VirtualBox website*. Retrieved from http://www.virtualbox.org/

Palopoli, L., & Cucinotta, T. (2007). Feedback scheduling for pipelines of tasks. In *Proceedings of the 10th Conference on Hybrid Systems Computation and Control*.

Parallels Holdings Ltd. (n.d.). *Parallels virtuozzo containers*. Retrieved from http://www.parallels.com/ products/ virtuozzo/ lib/ download/ wp/

Pratt, I. (2003). Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*.

Qumranet. (2010). *KVM - Kernal-based virtualization machine - White paper.*

Rajkumar, R., Juvva, K., Molano, A., & Oikawa, S. (1998). Resource kernels: A resource-centric approach to real-time and multimedia systems. *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*.

Rajkumar, R., Lee, C., Lehoczky, J. P., & Siewiorek, D. P. (1998). *Practical solutions for QoS-based resource allocation*.

Regehr, J., & Stankovic, J. A. (2001). Augmented CPU reservations: Towards predictable execution on general-purpose operating systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium* (RTAS 2001), May 2001, Taipei.

Saewong, S., Rajkumar, R., & Lehoczky, J. P. Klein & M. H. (2002). Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*.

Shankaran, N., Koutsoukos, X. D., Schmidt, D. C., Xue, Y., & Lu, C. (2006). Hierarchical control of multiple resources in distributed real-time and embedded systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*.

Shin, I., & Lee, I. (2004). Compositional real-time scheduling framework. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium* (pp. 57–67).

Srinivasan, A., & Baruah, S. (2002). Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, *84*(2), 93–98. doi:10.1016/S0020-0190(02)00231-4

Steere, D., Shor, M. H., Goel, A., & Walpole, J. (2000). Control and modeling issues in computer operating systems: Resource management for real-rate computer applications. In *Proceedings of 39th IEEE Conference on Decision and Control*.

Stoica, I., Abdel-Wahab, H., Jeffay, K., Sanjoy, K., Baruah, J. E., Gehrke, C., & Plaxton, G. (1996). A proportional share resource allocation algorithm for real-time, time-shared systems. *Proceedings of the IEEE Real-Time Systems Symposium*.

Tokuda, H., & Kitayama, T. (1993). Dynamic QoS control based on real-time threads. *NOSSDAV'93: Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video.*

VMware Inc. (2009). *Protecting mission-critical workloads with VMware fault tolerance*. Retrieved from http://www.vmware.com/ resources/ techresources/ 1094

VMWare Inc. (2010). *VMWare media resource center - Company milestones*. Retrieved from http://www.vmware.com/ company/ mediaresource/ milestones.html

Ward, M., & Audsley, N. (2002). Hardware implementation of programming languages for real-time. *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*.

Wust, C. C., Steffens, L., Bril, R. J., & Verhaegh, W. F. J. (2004). QoS control strategies for high-quality video processing. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*.

KEY TERMS AND DEFINITIONS

Quality of Service (QoS): Summary of mechanisms used to deliver an agreed service level.

Real-Time: Real-Time systems guarantee a maximum response time. Responses become worthless after the set deadline.

Scheduling: Determining when, i.e., the time interval, and where, i.e. on which processor, a job is executed.

Service Level Agreement (SLA): Contract defining the level at which a service is to be provided. It defines parameter s such as maximum delay and jitter, guaranteed bandwidth, and service availability.

Temporal Interference: Influence the execution of a virtual machine has on the timing of execution of another virtual machine on the same host.

Virtualization: Decoupling physical from logical entities. Multiple physical entities may be aggregated to form a single logical entity or a single physical entity may provide multiple logical entities.