2nd International Workshop on

Real-time and distributed computing in emerging applications

Co-located with IEEE RTSS

Vancouver, Canada

December 3rd, 2013

Marisol GARCIA VALLS and *Tommaso CUCINOTTA* Editors

Sponsors

Edited by *Marisol GARCÍA VALLS and Tommaso CUCINOTTA*

ISBN-10: 84-616-7680-4

ISBN-13: 978-84-616-7680-4

December 3rd, 2013

Available on-line:

http://hdl.handle.net/10016/17914

http://e-archivo.uc3m.es/handle/10016/17914

http://retis.sssup.it/reaction2013/REACTION-2013-Proceedings.pdf

This page has been left intentionally blank.

Message from the General Chairs

A number of challenges to achieve temporal predictability are faced by distributed applications given the complexity and scale of the current and upcoming domains. Approaches to enhance the traditional scheduling-centered focus of real-time research are needed since it is no longer possible to work solely on the assumptions of highly predictable execution platforms.

The germ of the first edition of the REACTION workshop initiated in 2012 was the idea of providing a forum for presenting novel contributions to merge real-time with the new computing paradigms and emerging applications that are intensive in the use of distribution.

In this second edition of REACTION 2013, we persist in our aim to bring together researchers from the real-time and the distributed systems communities to cross fertilize and provide fresh, novel, and (why not!) risky approaches that may open the road to new efficient solutions. We move on with our exploratory approach aiming at attracting the presentation and discussion of ideas of researchers working on distributed real-time systems for the next-generation applications. Contributions on both practical and theoretical aspects applied to the integration of real-time support in the new computation paradigms and emerging applications emphasizing aspects of real-time support for flexibility and system dynamics have been attracted.

The General Chairs of REACTION 2013 would like to thank all authors, contributors, and reviewers. They have shown that there is a real interest for the idea that we had in mind since the first edition. Also, we would like to thank the members of the Programme Committee for their support and help in making this event a reality. We would also like to thank Scuola Superiore Sant'Anna for their support in hosting the REACTION'13 website. Last but not least, we want to thank the RTSS Organizing Committee with special mention to the Workshops Chair for his outstanding support.

The REACTION 2013 General Chairs

Marisol GARCIA VALLS and Tommaso CUCINOTTA

This page has been left intentionally blank.

Keynote talk

Adaptive Resource Allocation in the Cloud

Srikanth Kandula, Microsoft Research, Redmond, USA

Carefully allocating resources can improve throughput, lower latency and offer more predictable service. In this talk, I will present three recent examples and point out future directions.

With SWAN, we show that given responsive networks and responsive applications adapting who gets to send how much, when, and along which network paths can improve network utilization without losing out on business priorities. We show how SWAN can be incorporated into the wide-area network of enterprises that have a global datacenter footprint. With Kwiken, we show how to improve the tail latency of datacenter services which are built as workflows over many components by appropriately allocating additional resources across the various stages in the workflow. Interestingly, we also cast incompleteness (i.e., returning partial results) as a resource and show that small amounts of incompleteness can improve latency by a lot. Finally, with RoPE, we show how execution plans for jobs in big data clusters can improve given additional information about properties of the user code, data and how the code and data interact. We also describe a system that extracts such properties at scale.

Biography

Srikanth Kandula is a Researcher at Microsoft Research. His research interests span many aspects of networked systems including datacenters, network management, diagnosis, applied statistical inference and security. He has published over 15 papers in top-tier venues such as SIGCOMM, NSDI, MobiSys and SIGMOD. He is a winner of the NSDI best student paper award (2005). Many of his research artefacts have been widely adopted in Windows and Microsoft's cloud infrastructure. He obtained his Ph. D. from the Massachusetts Institute of Technology (2008).

This page has been left intentionally blank.

Table of contents

Session A – Middleware and distributed systems

Experimental evaluation of the real-time performance of publish-subscribe middlewares 1
Tizar Rizano, Luca Abeni and Luigi Palopoli

Towards the integration of data-centric distribution technology into partitioned embedded systems 7
Héctor Pérez and J. Javier Gutiérrez

Benchmarking communication middleware for cloud computing virtualizers 13
Marisol García-Valls, Pablo Basanta-Val and Rosbel Serrano-Torres

Session B – Scheduling in distributed systems and multiprocessors

A feasible configuration of AFDX Networks for Real-Time Flows in Avionics Systems 19
Dongha An, Hyun Wook Jeon, Kyong Hoon Kim and Ki-Il Kim

Task partitioning and priority assignment for hard real-time distributed systems 25
Ricardo Garibay-Martínez, Geoffrey Nelissen, Luis Lino Ferreira, Luís Miguel Pinho

Session C – Cloud computing

Run-time support for real-time multimedia in the cloud 31
Tommaso Cucinotta, Karsten Oberle, Manuel Stein, Peter Domschitz and Sape Mullender

Resource management for service level aware cloud applications 47
Cristian Klein, Martina Maggio, Karl-Erik Arzén and Francisco Hernández

This page has been left intentionally blank.

Experimental Evaluation of the Real-Time Performance of Publish-Subscribe Middlewares

Tizar Rizano, Luca Abeni, Luigi Palopoli
Dipartimento di Scienza e Ingegneria dell'Informazione
University of Trento, Trento, Italy
tizar.rizano@unitn.it, luca.abeni@unitn.it, luigi.palopoli@unitn.it

***Abstract*—The integration of the complex network of modules composing a modern distributed embedded systems calls for a middleware solution striking a good tradeoff between conflicting needs such as: modularity, architecture independence, re-use, easy access to the limited hardware resources and ability to respect real–time constraints. Several middleware architectures proposed in the last years offer reliable and easy to use abstractions and intuitive publish-subscribe mechanism that can simplify system development to a good degree. However, a complete compliance with the different requirements of assistive robotics application (first and foremost real–time constraints) remains to be investigated. This paper evaluates the performance of these solutions in terms of latency and scalability.**

I. Introduction

The recent developments in sensing and battery technologies and in embedded computing devices are creating the premises for the development of low cost robotic applications for a consumer market. The ever-increasing presence of robot vacuum cleaners in our homes, of robotic toys amusing our children, of robotic drones shooting impressive pictures from surprising points of view are witnesses of a clear market trend. At the forefront of this movement are robots created to assist older adults or people with different disabilities. One of the basic needs that can effectively be addressed by assistive robots is personal mobility.

These embedded systems integrate several modules and rely on different types of sensors that convey information on the surrounding environment. For example, they can use video sensors to detect moving objects or obstacles, or can use gyroscopes encoders, 3D cameras and RFID readers for localisation purposes. The same level of complexity is on the software architecture, that can include modules for video-analysis, mission planning, short term planning and control. All these services might interact with other components such as a geo spatial database that stores relevant information about the environment (in this case, the geo spatial database maintains a consistent description of the environment, where each model inserts additional information layers).

The integration of this complex network of modules calls for a middleware solution striking a good tradeoff between conflicting needs such as: modularity, architecture independence, re-use, easy access to the limited hardware resources and real–time constraints.

Several middleware architectures proposed in the last years offer reliable and easy to use abstractions and intuitive publish-subscribe mechanism that can simplify the development of complex robotic applications to a good degree. Examples are OpenDDS[1], which implements a standard proposed by the Object Management Group[1], ZeroMQ [2], which implements a publish-subscribe paradigm to support concurrent programming over socket connections using a publish-subscribe paradigm and is freely available[2], and ORTE [3], which implements a publish-subscribe mechanism over a real–time Ethernet connection (in particular, it is compliant with the RTPS - Real-Time Publish-Subscribe - protocol).

The three mentioned solutions have different reasons of interest: OpenDDS builds on top of the decennial experience made by the CORBA community and offers powerful abstractions, ZeroMQ is extremely lightweight and potentially interesting for its easy adaptation to embedded architectures, and ORTE is a product has been developed for a special care for its real–time performance.

Based on some previous experience [4], this paper evaluates the performance of the three middlewares in terms of latency, scalability, and communication throughput. This comparison will be used as a cornerstone for the development of a reliable software architecture for the DALi cognitive walker (cWalker), an embedded device designed to assist adults with non-severe cognitive abilities in the navigation of complex and crowded environments (e.g., an airport or a mall), which challenge the sense of direction and generate anxiety. However, this work is not limited to the cWalker, but is aimed at increasing the diffusion of real–time middlewares in a large class of robotic applications.

The rest of the paper is organised as follow. Section II offers a high level overview of the case study. Section III, shortly describes the three middleware analysed in the paper and compares their features. Section IV, reports the experimental results on the performance comparison between the

[1]http://www.opendds.org
[2]http://www.zeromq.org

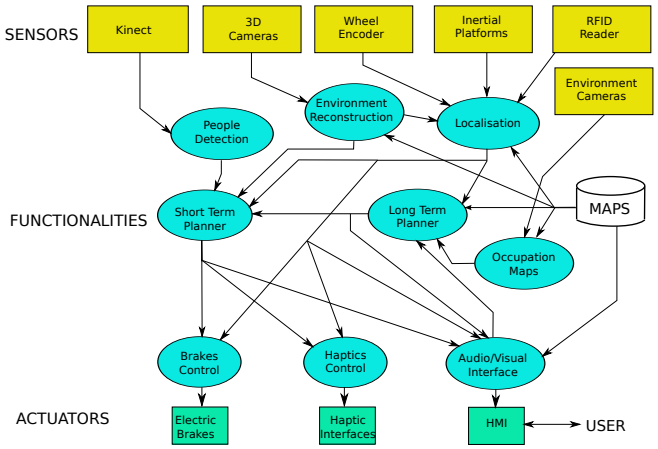


Figure 1. Simplified functional scheme of the DALi cWalker.

three different alternatives. Finally, Section V, presents some conclusions and a short discussion of future work directions.

II. Case–Study

An important motivational example for this work has been offered to us by a cooperative European project [3] coordinated by the University of Trento. The objective of the project is the development of a robotic assistant to help older adults with emerging cognitive impairments navigate large and challenging environments (e.g., a shopping mall, or an airport). Because the main focus of the project is to compensate for cognitive deficiencies, the assistant is called cWalker (cognitive walker). A simplified scheme of the most important functionalities of the cWalker is shown in Figure 1. The cWalker prompts the user for a sequence of target points in the environment that he/she wants to visit through a visual interface. The Long Term Planner finds the most convenient path using the map of the environment and the real–time information on the state of the place, which is acquired querying remote sensors (e.g., the surveillance cameras). When the users starts to move, the walker guides her/him along the path using electro-actuated brakes [5], haptic interfaces and audio/video interfaces. The guidance requires a real–time localisation system which tracks the position of the cWalker while it moves. Along the way, the cWalker localises the user in the environment, detects anomalies and the motion of people in the surroundings and plans deviation from the planned path when required (e.g., to avoid accidents or such behaviours as could violate the social rules). These tasks are performed by a Short–Term planner.

A description of the different functionalities is beyond the goals of the present paper, and can be found in previous work [4].

[3]http://www.ict-dali.eu

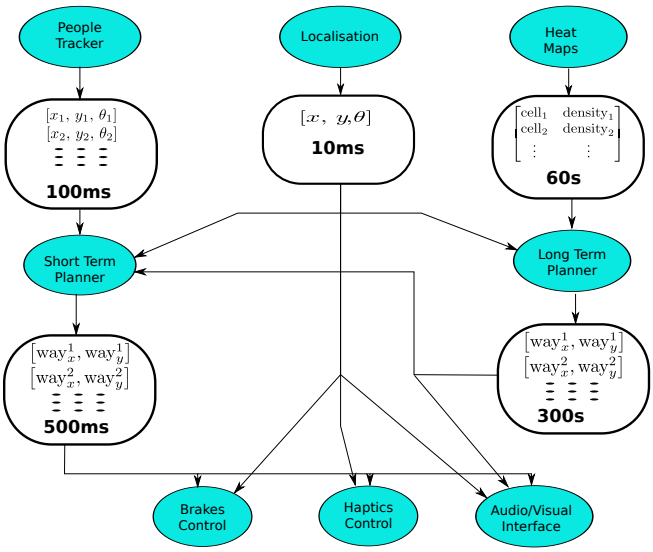


Figure 2. Publish-Subscribe architecture for some of DALi's components.

III. Publish-Subscribe Middlewares

The functional architecture described in Figure 1 suggests the following considerations:

1) Many of the components are re-usable across a wide family of applications and systems (e.g., the localisation module and the people tracker);
2) The computational demand and the physical constraints call for a distributed hardware implementation, in which the functionalities could be deployed in different nodes in different implementations or operating conditions (e.g., in response to a system failure);
3) The different components require varied expertise; the resulting development team is large and heterogeneous.

These requirements can be fulfilled by adopting a middleware infrastructure that implements publish-subscribe functionalities. Moreover, this solution simplifies the development and testing of the various modules, by permitting to decouple their development.

Figure 2 shows a possible implementation scheme for the communication between some of the modules. As an example, the people tracker publishes a sequence of positions and velocity of the people within the reach of the sensors with a periodicity of 100ms and this topic is subscribed to by the short term planner. The localisation module publishes a new position of the cWalker every 10ms and this information is used by various subscribers (at least those shown in the figure). Similarly in the graph one can read the topics published and subscribed to by other modules.

Since the cWalker modules are characterised by some real-time constraints (as shown in the previous example), the middleware implementing the publish-subscribe mechanism needs to be predictable and has to provide reasonable upper

bounds for the communication latencies without compromising the throughput. Hence, the middleware has to be explicitly designed to support real-time communications. While the idea of real-time publish-subscribe communication is not new [6], a systematic comparison of multiple *open-source* alternatives is still missing.

The Object Management Group (OMG) published various standards regarding real-time data exchange based on a publish-subscribe protocol. In particular, the Data Distribution Service (DDS) standard defines a service for distributing application data between tasks (in distributed applications), and the Real-Time Publish-Subscribe (RTPS) standard defines an application-level protocol based on UDP/IP, which can be used for the real-time communications required by DDS.

The DDS specification defines both an application level interface for a service implementing the publish-subscribe functionalities (in real-time systems) and an additional layer that allows distributed data to be shared between applications based on DDS. The first interface (Data-Centric Publish-Subscribe - DCPS) is in charge of efficiently delivering the proper information to the proper recipients (according the publish-subscribe) and introduces a *global data space* to be used by applications for exchanging data.

The second part of the standard (Data-Local Reconstruction Layer - DLRL) is a higher level software layer based on DCPS and uses it to construct local object models on top of the global data space.

DDS does not specify a specific "wire protocol" to be used for data exchange and control, hence different DDS implementations can use different (and incompatible) protocols, being them TCP-based, UDP-based, or something different (for example, 2 modules running on the same node can communicate through shared memory to improve the performance).

RTPS is a possible wire protocol to be used by DDS (technically speaking, it is an application-level protocol, generally based on UDP). The RTPS protocol has been designed focusing on real-time requirements, hence it allows to trade the reliability of message delivery for low latencies. As a result, it often implements real-time communications on top of unreliable and connectionless transport protocols such as UDP (although TCP can also be used - see OpenDDS below). The protocol supports publication and subscription timing parameters and properties to allow some performance vs reliability trade-offs.

When using DDS, a publisher and a subscriber communicate by writing/reading data identified by two parameters: *topic* and *type*: the topic is a label that identifies each data flow while the type describes the data format.

To provide good real-time performance (and to properly scale, without having the communication latency affected by the number of publishers or subscribers), DDS and RTPS do not rely on an active service that receives messages from the publishers and forwards them to the proper subscribers. Instead, peer-to-peer connections between each publisher and the interested subscribers are created, based on a naming service that can be provided by some dedicated daemon.

Finally, DDS provides automatic data serialisation through an Interface Definition Language (IDL) compiler, so that components running on different architectures can easily interoperate and communicate (notice, however, that this feature is not strictly needed in the DALi context, since the distributed architecture is based on uniform nodes).

One of the goals of this evaluation is to quantify the overhead (if any) introduced by the various DDS and RTPS abstractions, in order to understand their costs and their benefits. Hence, three different middlewares (ranging from one that is fully compliant with DDS to one that is not compliant with any standard) have been considered: OpenDDS, ORTE, and ZeroMQ.

OpenDDS is fully compliant with the DDS standard forces to use the IDL compiler to serialise the data to be exchanged. ORTE is less flexible, but still implements the RTPS protocol (and is explicitly focused on respecting real-time constraints). Finally, ZeroMQ is not compliant with any specific standard, does not provide a naming service, but relies on simplicity to provide good performance. Hence, comparing the three middlewares allows to evaluate the cost and the benefits of the various features described in the standards and to estimate the overhead that the various features and abstractions might introduce. In more details:

OpenDDS
: is an implementation of DDS v1.2 using RTPS as a "wire protocol" (according to the DDS-RTPS standard v2.1). Both UDP and TCP can be used as a transport protocol below RTPS. It is implemented using the C++ language and is based on CORBA (using ACE/TAO) for the naming and discovery service and for serialising the data (through the TAO IDL). This allows OpenDDS to provide cross platform portability and to easily implement the DCPS layer;

ORTE (the Open Real-Time Ethernet)
: is a lighter implementation of the RTPS protocol which does not rely on external software and directly implement RTPS using UDP sockets. Serialisation can be performed directly by the application. It is implemented using the C language;

ZeroMQ
: is an open source based messaging library implemented in C++ providing support for the publish-subscribe communication paradigm over TCP. Serialisation is not considered. It is not compliant with any standard, and does not provide any kind of naming service (which is then application's responsibility). It exports an object-oriented API with bindings for various languages e.g. C, C++,

python and Java.

IV. Performance Evaluation

The three middlewares have been compared by evaluating their performance in terms of both worst case and average real-time latencies.

This evaluation has been performed by using some test programs implementing publish-subscribe communication, and using a setup similar to the one described in Figure 2.

Since the specific middleware that will be used in the DALi walker has not been decided yet (but only the needed features have been identified), an abstraction layer providing the needed publish-subscribe functionalities has been developed. Such an abstraction layer exports a simplified API that allow to create publishers and subscribers, publish and receive topics, and perform all the operations needed by the various DALi modules.

In particular, the abstraction layer is written in C++ and its API is composed by:

- A class modelling global data space abstraction, where data is published and received by the subscribers;
- A class modelling a Publisher. This class can be instantiated once a global data space has been defined, and can publish a topic on such a data space;
- A class modelling a Subscriber. Similarly to the publisher class, this class can be instantiated only once a global data space has been defined, and receives messages concerning a specified topic from such a data space.

The global data space class only provide a constructor, a destructor, and two methods to create a Publisher or a Subscriber. When creating a Publisher, it is possible to specify a name for the topic it publishes; the Publisher class then provides a `publish()` method that allows to send messages for this topic. When creating a Subscriber, it is possible to specify the name of the topic to subscribe to; the Subscriber class then provides a `register_callback()` method that allows to specify a callback to be invoked when a message for the specified topic is received.

The C++ classes then hide all of the implementation details (and the middleware API), allowing to write code using the publish-subscribe paradigm without relying on a specific middleware. The abstraction layer currently supports the three middlewares considered in this paper, but extending it to other middlewares based on the publish-subscribe paradigm should be simple.

Some preliminary experiments measured the performance of the middleware without considering the effects of the network (by running the experiments on a single node) and revealed that ORTE seems to perform slightly better than the other middlewares when only few subscribers are active, but ZeroMQ scales better [4]. In any case, on an Intel i7 CPU running at $2.8GHz$ the worst-case measured latency was smaller than $1ms$, for all the middlewares.

In this paper, the experiments have been performed using a setup that is more similar to the DALi hardware and software architecture. First of all, the embedded boards that will probably be used in the DALi cWalker (pandaboards[4], based on an OMAP4460 - powered by an ARM core running at $1Ghz$) have been used. Moreover, the experiments are performed on two identical pandaboards connected via fast ethernet switch (100 Mbps); hence, network effects have been accounted for in the experiments. The two boards run Ubuntu 12.04 with the 3.2.0 Linux kernel.

A first set of experiments, still based on the simple test programs used in the previous paper, compare the real-time performance of the three middlewares by measuring the latency between the generation of a message (from the publisher) and its arrival to the subscribers - this will be referred as "publish-subscribe latency". With respect to the previous experiments, the ones reported here are based on the pandaboard setup described above. First, some "single node" experiments (similar to the previous ones) have been run, and then the measurement have been repeated with the publisher running on one board and the subscribers running on the other one. As in the previous experiments, the middleware abstraction layer has been used to easily repeat the same tests with different middlewares.

The publisher is implemented as a single-threaded process scheduled with `SCHED_FIFO` and the maximum real-time priority. Each subscriber (maximum 4 subscribers) is also a high priority (`SCHED_FIFO`, maximum real-time priority) process. However, the process is multi-threaded, since all of the tested middlewares create at least two threads for each subscribers: main thread and the subscriber listener thread. For OpenDDS, there is an extra thread that run its ORB and several threads for non-CORBA transport IO. OpenDDS and ORTE are configured to use UDP as their transport protocol. However, ZeroMQ is configured to use TCP since UDP is not officially supported.

Figure 3 reports the results (worst-case and average latencies as a function of the number of subscribers) obtained when running publisher and subscribers on the same node. Respect to the results obtained on the x86-based PC, the worst-case latencies are about 10 times larger, and the ORTE behaviour is slightly worse than the ZeroMQ one (in the previous experiments, ORTE behaved better than ZeroMQ for small numbers of subscribers, but ZeroMQ scaled better).

Figure 4 reports the results of the same experiment executed in a distributed environment (publisher and subscribers on 2 different nodes). It is immediately possible to notice that the latencies increase even more, and only ZeroMQ stays below 10 ms in both average and worst case latencies for all the numbers of subscribers. Again, confirming the result obtained in [4] ORTE performs well with a limited number of subscribers while ZeroMQ scales better than the

[4] http://www.pandaboard.org

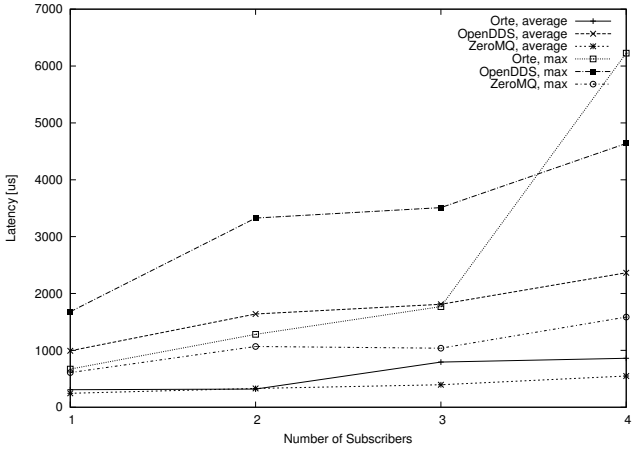


Figure 3. Single node Publisher/Subscriber latency as a function of the number of subscribers.

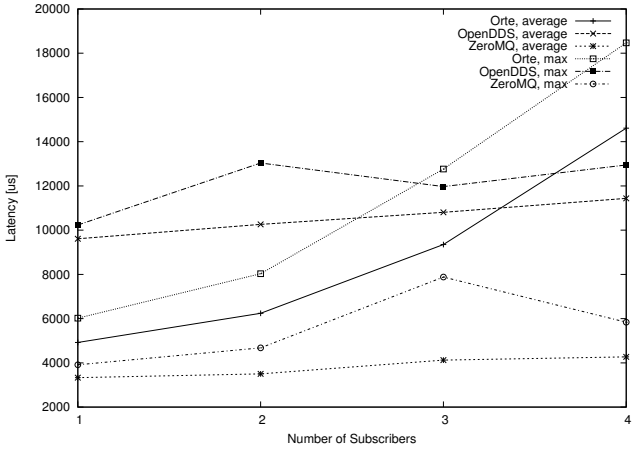


Figure 4. Multi node Publisher/Subscriber latency as a function of the number of subscribers.

other middlewares even in the distributed scenario.

Finally, Figure 5 reports the latencies as a function of the message size, showing that the average latencies of all middlewares scale well with message size up to 1000 bytes.

After running the first experiments with a simplified test application, a more realistic test case based on Figure 2 has been used to compare the three middlewares. The test is composed by 8 processes emulating the 8 software modules that will run on the cWalker: the *People Tracker* (PT), the *Localization module* (LOC), the *Heat Maps* (HM), the *Short Term Planner* (STP), the *Long Term Planner* (LTP), the *Brakes Control* (BC), the *Haptics Control* (HC) and the *Audio Visual Interface* (AVI). All the modules are modelled as periodic real-time tasks running with the periods indicated in Figure 2, subscribing to some topics, and eventually producing messages at each activation.

Each task/software module is statically assigned to a pandaboard, and different ways to distribute the tasks have

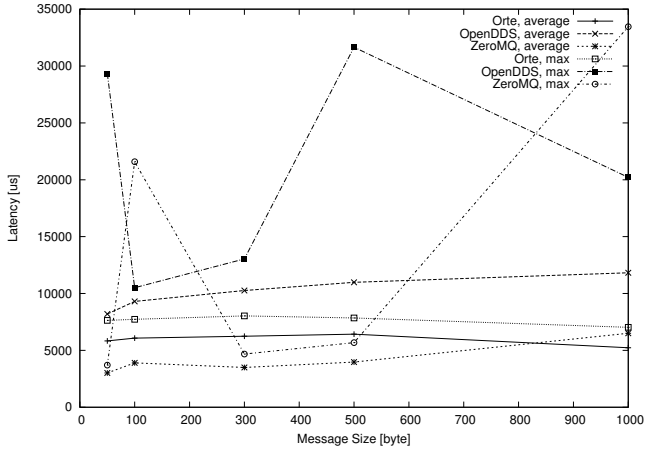


Figure 5. Multi node Publisher/Subscriber latency as a function of the message size

Mapping	protocol	max	avg	stdev
1	ZeroMQ	1740	523.41	183.68
	ORTE	7599	712.39	260.39
	OpenDDS	6135	2016.79	470.46
2	ZeroMQ	6752	2368.42	427.55
	ORTE	10170	4563.44	832.93
	OpenDDS	11268	4952.68	680.27
3	ZeroMQ	7851	3720.68	680.27
	ORTE	11940	5092.19	410.39
	OpenDDS	11482	6179.72	295.92

Table I
LATENCY IN MICROSECONDS

been tested. In particular, the results obtained with three different mappings of modules to embedded boards will be reported:

- **Mapping 1**: All modules run on pandaboard 1
- **Mapping 2**: The AVI, HM, and LTP modules run on pandaboard 1 while BC, HC, PT, LOC, and STP run on pandaboard 2
- **Mapping 3**: The AVI module runs on pandaboard 1 while all the other modules (BC, HC, PT, LOC, STP, HM, and LTP) run on pandaboard 2.

The worst-case and average latencies measured the output of the AVI module are reported in Table I. This set of experiments show the effect of distributed processes on the performance of the middlewares. The average latencies of all middlewares stay below the minimum period of the modules (10 ms). However, the worst case latencies of all middlewares except ZeroMQ are above the minimum period.

V. CONCLUSIONS

This paper presents the performance evaluation of three open-source publish-subscriber middlewares. The evaluation focuses on their real-time performance, to identify the solution that best suits the needs of modern robotic

applications based on distributed embedded architectures. The experimental setup was designed taking inspiration from an existing robotic application.

Based on the result of the experiments, ZeroMQ is shown as the most suitable middleware for DALi application. Although the average latencies of both ORTE and OpenDDS are below the minimum period required by DALi application, their worst case latencies is above it. However, Their latencies remain below 7 ms for 99% of the time.

The goals of future investigations are manifold. One of the most important is to extend the analysis to other middleware solutions explicitly developed for robot applications such as ROS [7] and OROCOS [8].

REFERENCES

[1] OMG, "Data distribution service for real-time systems – version 1.2," The Object Management Group, Tech. Rep., 2007.

[2] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. O'Reilly, 2013.

[3] P. Smolik, Z. Sebek, and Z. Hanzalek, "Orte–open source implementation of real-time publish-subscribe protocol," in *Proc. 2nd International Workshop on Real-Time LANs in the Internet Age*, 2003, pp. 68–72.

[4] T. Rizano, L. Abeni, and L. Palopoli, "Middleware for robotics in assisted living: A case study," in *Proceedings of the 15th Real-Time Linux Workshop*, Lugano, Switzerland, October 2013.

[5] D. Fontanelli, A. Giannitrapani, L. Palopoli, and D. Prattichizzo, "Unicycle steering by brakes: a passive guidance support for an assistive cart," in *Proceedings of the 52nd IEEE Conference on Decision and Control*, Firenze, Italy, December 2013.

[6] R. Rajkumar, M. Gagliardi, and L. Sha, "The real-time publisher/subscriber inter-process communication model for distributed real-time systems: design and implementation," in *Proceedings of the 1st Real-Time Technology and Applications Symposium (RTAS95)*, 1995, pp. 66–75.

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[8] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528.

Towards the integration of data-centric distribution technology into partitioned embedded systems

Héctor Pérez
Computers and Real-Time Group
Universidad de Cantabria
Santander, SPAIN
perezh@unican.es

J. Javier Gutiérrez
Computers and Real-Time Group
Universidad de Cantabria
Santander, SPAIN
gutierjj@unican.es

Abstract—**This work proposes an architecture to enable the use of data-centric real-time distribution middleware in partitioned embedded systems based on a hypervisor. Partitioning is a technique that provides strong temporal and spatial isolation, thus allowing mixed-criticality applications to be executed in the same hardware. The proposed architecture not only enables transparent communication among partitions, but it also facilitates the interconnection between partitioned and non-partitioned systems through distribution middleware. Preliminary results show that hypervisor technology provides low overhead and a reasonable trade-off between temporal isolation and performance.**

Keywords—distributed systems; middleware; hypervisor; DDS; real-time systems.

I. Introduction[1]

Partitioning is a widespread technique that enables the execution of multiple applications in the same hardware platform with strong temporal and space isolation, thus allowing the coexistence of mixed-criticality applications, which fulfils their different requirements (i.e. integrity, security, timing, etc.). Although partitioned systems were initially conceived for safety-critical contexts and do not traditionally contemplate the use of distribution middleware because of its complexity, this technique is becoming more and more popular and it is starting to be applied in a heterogeneous set of emerging applications [1].

The use of middleware technology can provide a set of services that may be of interest for partitioned systems, such as location transparency, abstraction of network services, communication management or interoperability. As part of modern model-driven software development techniques, it also may help to resolve key challenges in the development and validation of distributed systems [2] [3]. Over the last years, the Data Distribution Service for Real-Time Systems (DDS) standard [4] has been attracting an increasing interest within the industry due to its flexibility and decoupling capabilities, along with a rich set of *Quality of Service* (QoS) parameters. These features make this standard suitable for the development of distributed systems with real-time requirements [5][6].

1. This work has been funded in part by the Spanish Government and FEDER funds under grant number TIN2011-28567-C03-02 (HIPARTES).

Our concern is to enable partitioned systems to take advantage of common real-time distribution middleware in several scenarios where a high level of criticality is not required. Under these conditions, important design objectives for partitioned systems include software reuse or interoperability between partitioned and non-partitioned systems. Both objectives can be fulfilled by integrating distribution middleware into partitioned systems as shown in [7], which presents an early experience dealing with RT-CORBA [8] and Ada DSA [9] standards. Furthermore, there is an initial attempt to extend DDS with a safety-critical profile [10][11] suitable for partitioned systems such as those defined by ARINC-653 (Avionics Application Standard Software Interface) [12], which proposes this standard as a suitable candidate to interconnect the next-generation of partitioned distributed real-time systems.

Therefore, this paper proposes a system architecture that integrates the use of distribution middleware based on the DDS standard within XtratuM [13], which is an ARINC-653-like hypervisor especially designed for real-time embedded systems. Additionally, a prototype has been developed in order to provide a performance analysis that estimates the overhead incurred when using the proposed architecture. The trade-off between performance and temporal/spatial isolation capabilities is also analysed.

To the best of our knowledge, few research papers have dealt with the merging of DDS and virtualization technology. For instance, the authors in [14] use DDS to interconnect virtual resources on heterogeneous hypervisors. Furthermore, the impact of using DDS in a general-purpose virtualized scenario is addressed in [15]. However, our work differs from these in the target systems, as XtratuM is specially designed to be used in scenarios with hard real-time requirements, in which safety-critical features can be also considered.

This document is organized as follows. Section II introduces the basic characteristics of XtratuM and the DDS standard. The architecture for integrating DDS middleware with XtratuM is proposed in Section III. Section IV describes a potential application as a proof of concept, while Section V evaluates the performance of the proposed architecture. Finally, Section VI draws the conclusions.

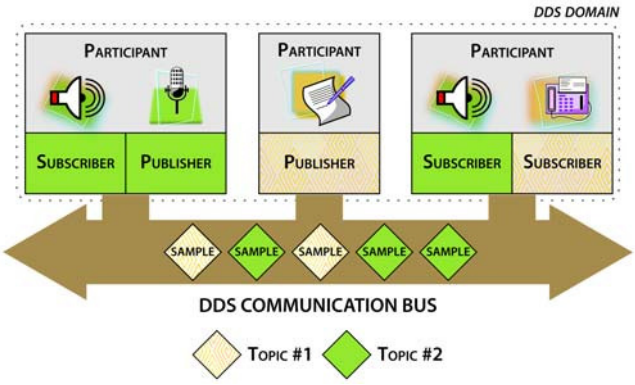


Fig. 1 DDS architecture

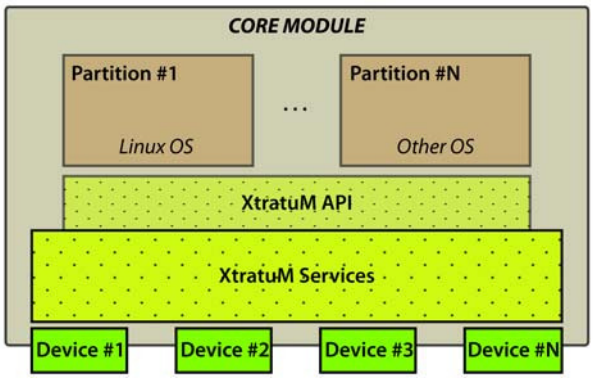


Fig. 2 XtratuM architecture

II. BACKGROUND

A *Overview of DDS*

The Data Distribution Service (DDS) standard defines a data-centric distribution middleware that supports the development of distributed real-time systems [16] by including a wide set of configurable parameters to provide different degrees of QoS. The standard is based on the publisher-subscriber paradigm, where *publishers* and *subscribers* communication entities respectively write (produce) and read (consume) data. All the communication entities that share compatible QoS parameters may be grouped in *participants* of a *domain*, and only entities belonging to the same domain can communicate.

To enable the communication among entities, publishers require to declare their intent to publish a specific *topic* (i.e. the data type to share), while subscribers require to register their interest in receiving particular topics. The example in Figure 1 illustrates a distributed system which consists of three participants in a single domain and two topics. Both topics have a single publisher in charge of generating new data samples. However, successive updates for topic # 1 will only be received by one subscriber, whereas new samples for topic # 2 will be received by two subscribers.

B *Overview of XtratuM*

XtratuM [13] is an open source hypervisor with capabilities to meet real-time and integrity requirements. Although it does not follow a specific standard, its design follows the philosophy of the ARINC-653 avionics standard [12]. This specification defines the interface of a partition-based operating system that allows multiple applications to execute in the same hardware platform, while maintaining time and space isolation. The general architecture of a system using XtratuM is shown in Figure 2, where the term partition represents one or several applications executing over a bare machine or an operating system. Each partition is allocated one or several dedicated time windows during which it may execute and thus multiple partitions can be concurrently executed on the same core module (a hardware platform with one or more processors or cores). Among the facilities provided by XtratuM are the virtualization of the basic resources of the system (clocks, timers, memory, interrupts, etc.) and specific communication services.

Two different and complementary communication services are defined in XtratuM: the ARINC-like communication ports [12] or the XMIO communication service based on Virtio [17]. The former was designed to enable communication in high-integrity systems (e.g., systems with static workload and pre-configured communication links), while the latter is aimed at non-critical software systems with some kind of timing requirements.

In XtratuM, the control and management of devices is left to partitions. To this end, XtratuM provides a configuration service to access the I/O ports, which must be configured at compilation time. I/O ports can belong to only one partition, which means that specific I/O partitions should be created when more than one partition needs to access a particular device. Furthermore, I/O partitions are responsible for implementing the device drivers so devices shared among several partitions should be managed in a special way, as described in the next section.

III. SYSTEM ARCHITECTURE

This section aims to explore the possible architectures that enable the use of data-centric distribution middleware in partitioned systems in which a hypervisor is used to manage the hardware. To guarantee the interoperability among non-critical open subsystems, our proposal will rely on the DDS distribution standard and the XMIO communication service. The analysis for more restrictive scenarios, which may require the use of the ARINC-like communication services and/or a reduced set of the DDS features, is left for future work.

As XtratuM does not implement drivers at the hypervisor level, sharing a device such as a network interface card (NIC) among multiple partitions should focus on handling

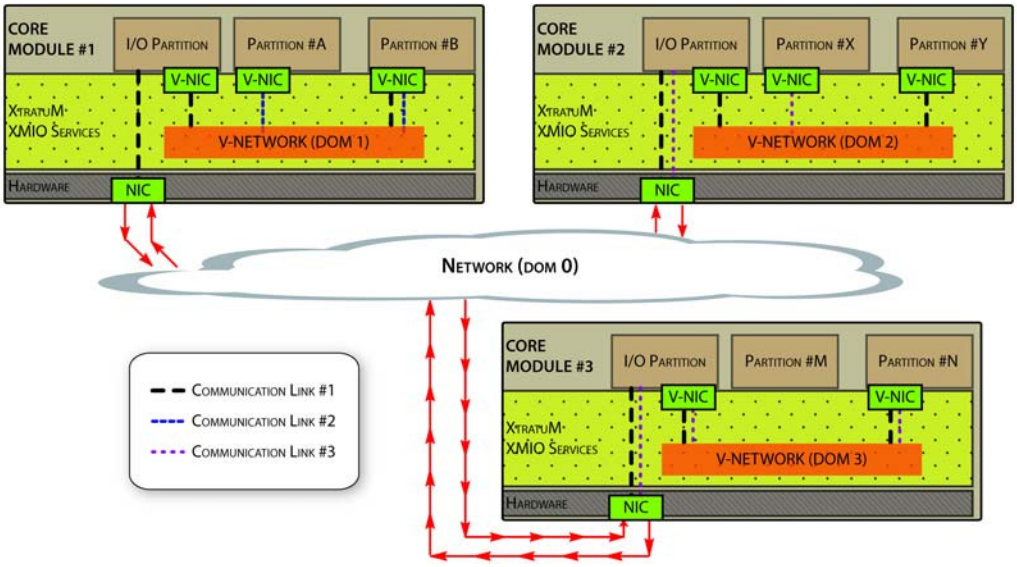


Fig. 3 Proposed architecture for integrating DDS with XtratuM

the contention in order not to compromise both space and time isolation capabilities. A common strategy is the use of an I/O partition that has exclusive access to the network card. Under this approach, the I/O partition is responsible for redirecting messages from the remaining partitions within the same core module to the communications network. To this end, two design strategies could be followed:

- *Designing an I/O partition exclusively aimed at forwarding messages.* In this case, messages are opaque to the I/O partition, and they would be routed through statically established connections. Therefore, each partition should know the destination of each communication link beforehand, which may not be suitable for open systems with variable workload.
- *Considering the use of DDS middleware in the I/O partition.* Thus, data-centric middleware will be responsible for performing routing transparently (e.g., based on topics). In this case, messages are not opaque and can be processed by the I/O partition. Moreover, this option may enable the use of different domains for inter- and intra-communication in core modules, as they may need to maintain certain information contained within.

Hence, each partition should implement data-centric middleware in order to provide distribution facilities such as location transparency, interoperability or connection management, and to facilitate data routing in the case of the I/O partition.

Figure 3 shows a system with three core modules following the proposed architecture for integrating data-centric middleware with a partitioned system using XtratuM. Communications between partitions, belonging or not to the same core module, are performed via DDS. As can be seen in the figure, each core module provides: (1) a virtual network (V-NETWORK) to enable the communication among partitions within the core module, which denotes a DDS domain; (2) a virtual network card (V-NIC) for each partition; and (3) an I/O partition, with exclusive access to the network card, which is responsible for routing the messages received by the underlying communication network, and which is part of another DDS domain. In this case, we have defined three communication links that interconnect partitions: link #1 defines one-to-many communications (i.e., one publisher and several subscribers); link #2 defines one-to-one communications within the same core module; and link #3 defines one-to-one communications between different core modules.

IV. USAGE SCENARIO: VIDEO-SURVEILLANCE SYSTEMS

This section describes a video-surveillance system as a proof of concept in which the use of the proposed architecture can be advantageous. Built-in video-surveillance applications will probably become common in the near future, for example in vehicles for recording unexpected situations (accidents, thefts, etc.). A key feature for this kind of systems resides in the reliability of the recording application, as it must keep recording data continuously, so it can benefit from strong isolation capabilities and can be executed together with other applications. In our example, a distributed application with multiple display monitors may request video captures from the recording application. The architecture for the proposed system is depicted in Figure 4 and it is composed of:

- One core module with two partitions: the *Video_Recorder* partition which is responsible for obtaining data from the attached video cameras and serving the requested video captures to other partitions, and the *Routing_Service* partition which is in charge of

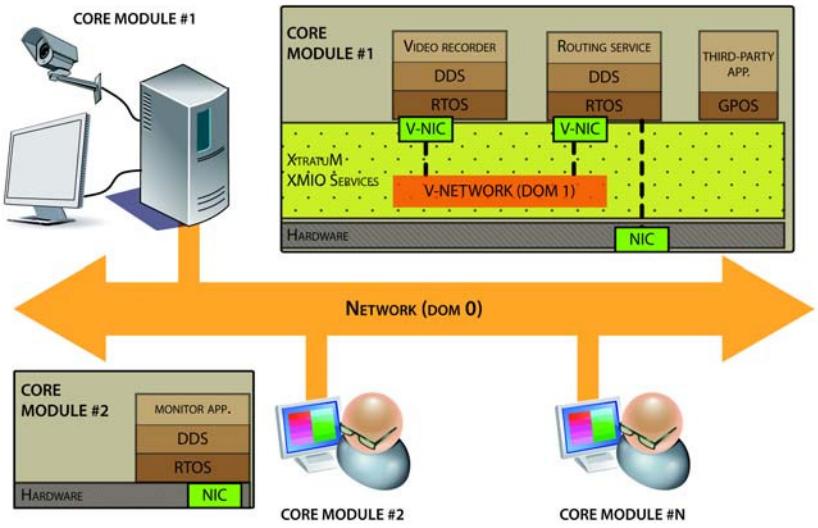


Fig. 4 Scheme of a video-surveillance system

routing the data from/to other core modules spread across the distributed systems.

- A variable number N of core modules that may request the current live video stream or a previous recording (i.e. the monitoring subsystem). These nodes or core module may or may not be partitioned systems.

The use of DDS enables the interoperability among the video recorder and the monitoring subsystems, regardless of whether they are partitioned or not. Furthermore, it also enables the interconnection among the partitions within the same core module (e.g., *Video_Recorder* and *Routing_Service* partitions). The *Routing_Service* partition also relies on DDS to control the information that flows in and out of the core module by providing data distribution between domains. Finally, third-party applications can be easily integrated into the system without compromising the security and data integrity of the *Video_Recorder* partition, as they are isolated in terms of space and time.

V. PERFORMANCE METRICS

This section aims to obtain preliminary performance metrics and assess the interoperability capabilities of using data-centric middleware in partitioned systems by simulating the video-surveillance scenario described in the previous section. In this evaluation, the distributed application consists of two nodes: the video recorder partitioned subsystem and one monitoring non-partitioned subsystem. The hardware platform consists of two single core 2.8 GHz nodes connected through an isolated Gigabit switch in which internal traffic has been disabled (for instance, network packets coming from the Spanning Tree or ARP protocols). We have adapted and integrated in a software platform: RTI Connext DDS[1] as distribution middleware, a fully pre-emptive Linux kernel 2.6.30.5 as the operating system and XtratuM as the hypervisor. Furthermore, a DDS add-on included in the RTI toolsuite[1] has been used to implement the routing service.

In the case of partitioned systems, the optimal configuration of partitions to maximize the processor's utilization is not a trivial problem, and it is even harder with inter-partitions dependencies. Thus, an I/O partition should be executed with sufficient regularity to fulfil the I/O requirements of other partitions. In our example, it is expected that the execution time of middleware operations will be similar to the ones associated with the routing operations, as both rely on DDS middleware. Hence, the video-surveillance application has been configured to have a dedicated time window of 800µs for the *Routing_Service*, and 700µs for the *Video_Recorder* partition, resulting in a scheduling plan repeated every 1,500µs.

The test will measure the execution time of a remote operation that publishes the requested video frames. We measure the operation carried out from the time when the request of a video capture is made until the image is returned. This operation is executed 10,000 times, and the average, maximum, and minimum times are estimated, together with the standard deviation and the 99th percentile (i.e., the value below which 99 percent of the measurements are found). To avoid additional overheads in the measurements, the test is executed without requiring network fragmentation (i.e., the payload is bounded to 1 kilobyte). The performance analysis includes two case-studies.

The first case study, which is called the *overhead test*, aims to estimate the overhead added by XtratuM when it is used as hypervisor. Three scenarios have been defined for this case study:

1. RTI-DDS toolsuite is available at http://www.rti.com

- *Network*, which estimates the temporal cost of using the network (transmitting and receiving a message of 1 kilobyte) by implementing the test over UDP in isolation.
- *Traditional DDS*, which measures the performance of the video-surveillance application using DDS over two non-partitioned nodes.
- *Single DDS Partition*, which measures the performance of the video-surveillance application when the node under analysis is partitioned with XtratuM. In this case, the core module only holds one partition that executes the application and has exclusive access to the network device. Therefore, this scenario estimates the overhead of using XtratuM.

The second case-study (*performance test*) evaluates the performance of the proposed system architecture, that is, with one partition dedicated to the I/O operations. To perform a fairer comparison, this case is contrasted with the traditional distributed application in which a routing service has been added. Therefore, two scenarios have been defined:

- *Traditional DDS with Routing*, which measures the performance of the video-surveillance application using DDS over two non-partitioned nodes. One of the nodes also executes a routing application to enable the communication between domains.
- *Partitioned DDS with Routing,* which measures the performance of the video-surveillance application when the proposed partitioned architecture is applied to one node. In this case, the core module holds two partitions: (1) the *Video_Recorder* partition, and (2) the *Routing_Service* partition to enable the communication between domains.

The results of the analysis for the *overhead test* are shown in Table 1. As can be observed, the DDS example adds a minimum overhead to the network test which makes it suitable for developing our approach, as it requires a lightweight middleware implementation in each partition. Likewise, the maximum overhead of using the distributed application on top of XtratuM is less than 60µs. Taking these metrics into account, it is shown that using hypervisor technology with data-centric middleware is highly efficient.

Table 1: Measurements of response times for the overhead test (in µsecs)

	Min	Avg	Max	Std	Per99
NETWORK	154	206	262	20	249
TRADITIONAL DDS	218	286	415	29	383
SINGLE DDS PARTITION	262	331	467	28	409

Table 2 shows the results of the measurements taken for the *performance test*, in which the proposed system architecture adds complexity by integrating a routing service into the distributed application. As shown in Table 2, the distributed operation for the DDS with routing scenario takes a maximum of 1,632µs, while this value is 4,157µs for the partitioned system. This variation in performance depends on the nature of the partitioned systems and their time window configuration (e.g., a network message received during the execution of the *Video_Recorder* partition has to wait until the next time window corresponding to the *Routing_Service* partition). In our example, we use a time window configuration that allows Linux partitions to be executed properly, as the optimization of time windows for this particular application is beyond the scope of this paper. In any case, the increase in the response times corresponds to a reasonable number of measurements for less critical applications (see the 99th percentile).

Table 2: Measurements of response times for the performance test (in µsecs)

	Min	Avg	Max	Std	Per99
TRADITIONAL DDS WITH ROUTING	662	764	1632	36	876
PARTITIONED DDS WITH ROUTING	1028	1858	4157	539	3346

To complete the study, an additional test has been carried out to evaluate the impact of the proposed architecture with different workloads. Figure 5 depicts the results obtained for the same experiment but using different image sizes. Similarly to the results obtained in Table 1 and Table 2, it is shown that the hypervisor adds a minimum overhead to the traditional DDS scenario regardless of the payload, and the maximum response times are appreciably higher for the partitioned system due to the inherent effect produced by the temporal partitioning. As a consequence of these results, a significant improvement is expected when using a

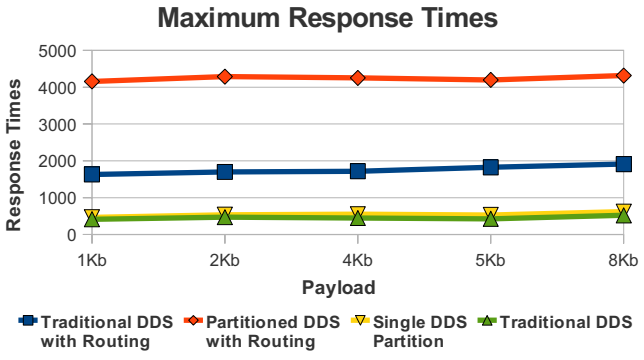


Fig. 5 Maximum response times for different image sizes (in µsecs)

multiprocessor approach that allows, for example, one core to be dedicated to communications, which could avoid the extra delays inherent to the time window configuration. This approach is planned for future work.

VI. Conclusions and Future Work

An increasingly important trend in many domains, such as the automotive, energy distribution or industrial control ones, is support for mixed-criticality applications within the same hardware platform. In this kind of applications, there is also a need to address the integration with the underlying communication subsystem. The proposed integration of DDS data-centric middleware into partitioned systems provides important benefits such as (1) the transparent invocation of services allocated in partitions, independently of whether they are in the same processor (or core) or in different ones; (2) the abstraction of network services which allows the application code to be simplified while maintaining it independent from the communication subsystem; and (3) interoperability between partitioned and non-partitioned systems, or between two or more heterogeneous partitions, e.g., with different levels of criticality or using different data representations (e.g., endianness).

As a consequence of the response times obtained in the performance analysis, it can be observed that the overhead of using data-centric middleware together with a partitioned system could be reasonable for a wide range of applications with soft real-time requirements. However, a significant improvement is expected when using the hypervisor technology adapted to multiprocessor systems, as it may partially mitigate the delays associated with the configuration of time windows. Anyway, it has been shown that this configuration is not a trivial problem and it represents a key step in the design of distributed applications with a partitioned architecture.

Although this integration can facilitate the use of partitioned systems with DDS, further investigation is required to fully determine which features of the standard can be applied, i.e., the applicability of some QoS configurations. Furthermore, it could be interesting to explore other approaches such as the use of the ARINC-like communication services for the incoming safety-critical profile of DDS.

References

[1] Multi-cores Partitioning for Trusted Embedded Systems (MULTIPARTES) European Project, Part of the 7th Framework Programme, http://www.multipartes.eu. 2013.

[2] A. Gokhale, K. Balasubramanian, A.S. Krishna, J. Balasubramanian, G. Edwards, G. Deng, E. Turkay, J. Parsons, and D.C. Schmidt, "Model driven middleware: A new paradigm for developing distributed real-time and embedded systems," Science of Computer Programming 73, pp. 39-58, 2008.

[3] K. An, T. Kuroda, A. Gokhale, S. Tambe, and A. Sorbini, "Model-driven Generative Framework for Automated OMG DDS Performance Testing in the Cloud," 12th International Conference on Generative Programming: Concepts & Experiences (GPCE), Indianapolis (USA), 2013.

[4] Object Management Group. Data Distribution Service for Real-time Systems. OMG Document, v1.2, formal/07-01-01. 2007.

[5] M. Ryll, and S. Ratchev, "Application of the Data Distribution Service for Flexible Manufacturing Automation," International Journal of Aerospace and Mechanical Engineering (2:3), pp. 193-200, 2008.

[6] F. Ben Cheikh, M.A. Mastouri, and S. Hasnaoui, "Implementing a Real-Time Middleware Based on DDS for the Cooperative Vehicle Infrastructure Systems," 6th International Conference on Wireless and Mobile Communications (ICWMC), Valencia, (Spain), 2010.

[7] H. Pérez, and J. J. Gutiérrez, "Experience with the integration of distribution middleware into partitioned systems," Proc. of the 17th International Conference on Reliable Software Technologies, LNCS, vol. 7896. Springer, 1-16. 2013.

[8] Object Management Group. Realtime Corba Specification. OMG Document, v1.2. formal/2005-01-04. 2005.

[9] Ada 2012 Reference Manual. Language and Standard Libraries - Intl. Standard ISO/IEC 8652:2012(E). 2012.

[10] R. Wahlin, and G. Hunt, "Towards a Safety Critical profile for DDS," Real-time and Embedded Systems Workshop, Arlington, VA (USA), 2009.

[11] R. Karoui, and A. Corsaro. "Real time Data Distribution for Airborne Systems," Workshop on Real-time, Embedded and Enterprise-Scale Time-Critical Systems, Washington DC, (USA), 2011.

[12] Airlines Electronic Engineering Committee, Aeronautical Radio INC. "Avionics Application Software Standard Interface". ARINC Specification 653-1. March, 2006.

[13] M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge, "Xtratum a hypervisor for safety critical embedded systems," Proc. of the 11th Real-Time Linux Workshop, Dresden (Germany), 2009.

[14] Y. Cho, J. Choi, and J. Choi, "An integrated management system of virtual resources based on virtualization API and data distribution service," Proc. of the ACM Cloud and Autonomic Computing Conference, New York (USA), 2013.

[15] R Serrano-Torres, M García-Valls, and P Basanta-Val, "Virtualizing DDS middleware: performance challenges and measurements," Proc. of 11th IEEE International Conference on Industrial Informatics, Bonchum (Germany), 2013.

[16] H. Pérez, and J. J. Gutiérrez, "On the schedulability of a data-centric real-time distribution middleware," Computer Standards & Interfaces (34:0), pp. 203-211, 2012.

[17] M. Masmano, S. Peiro, J. Sanchez, J. Simo, and A. Crespo, "IO Virtualisation in a Partitioned System," Proc. of the 6th Embedded Real Time Software and Systems (ERTS^2), Paris (France), 2012.

Benchmarking communication middleware for cloud computing virtualizers

Marisol García-Valls, Pablo Basanta-Val, Rosbel Serrano-Torres
Distributed Real-Time Systems Laboratory
Departamento de Ingeniera Telemática
Universidad Carlos III de Madrid
Av. de la universidad 30
28911 Leganés, Madrid, Spain
{mvalls,pbasanta}@it.uc3m.es

***Abstract*—Virtualization technologies typically introduce additional overhead that is specially challenging for specific domains such as real-time systems. One of the sources of overhead are the additional software layers that provide parallel execution environments which reduce the effective performance given by the infrastructure. This work identifies the factors to be analysed by a benchmark for performance evaluation of a virtualized middleware. It provides the set of benchmark tests that evaluate empirically the overhead and stability on a trendy communication middleware, DDS (Data Distribution System for Real-Time), which enables message transmissions via publisher-subscriber (P/S) interactions. Two different implementations, RTI and OpenSplice, have been analysed over a general purpose virtual machine monitor to evaluate their behavior on a client-server application. Obtained results have provided initial execution clues on the performance that a virtualized communication middleware like DDS can exhibit.**

I. Introduction

Communications middleware and virtualization technologies have been two main contributions to the development and maintainability of software systems. On the one hand, middleware brings in the capacity to abstract the low level details of the networking protocols and the associated specifics of the physical platforms (e.g. endianness, frame structure, and packaging, among others). This augments the productivity of systems development by easing the programmability and debugging. More recently, virtualization technologies have promoted a new technological trend that has fast penetrated different domains due to the benefits that it brings about: a) speed up of the customized system development and deployment to specific platforms; b) server consolidation and the subsequent savings on energy, etc. ; c) reducing maintenance and deployment costs and d) data availability any time and anywhere.

Communication middleware and virtualization technology originated for general purpose distributed applications, so initially in a different perspective from that of real-time environments where determinism is a key target. As science evolves and new applications are envisioned and engineered, real-time applications have progressively approached middleware and virtualization technologies, facing the problem of temporal predictability. The traditional focus of real-time and middleware has been significantly different. Networked real-time systems traditionally have focused on eliminating (or minimizing) the sources of unpredictability by direct programming of tasks in the real-time operating system or directly in the hardware platform itself and using controlled medium access protocols to develop real-time networks. Middleware has typically been implemented for distributed systems over non collision-free networks, and using software engineering techniques that introduce additional software layers aiming at easing programmability and interoperability. As a consequence, communications middleware has appeared as a black box, containing extra code that is difficult to analyse with sufficient level of detail and guarantees as required by some real-time applications.

Over the past decade, the OMG's DDS standard [1] (Data Distribution Service for Real-Time Systems) has appeared with considerable success for distributed soft real-time applications. DDS provides an asynchronous interoperability via a publish-subscribe paradigm that is data-centric. One of the success factors of DDS is that it provides quality of service (QoS) communication by means of specifying a collection of diverse QoS parameters. There are different realizations of the DDS standard that achieve different behaviors, mainly with respect to performance and to the specific set of implemented QoS parameters. In general, the level of temporal guarantees provided by different implementations varies depending on different factors such as the physical deployment, application type, and middleware communication paradigm and fine-tuning. There are not many public independent studies about the performance achieved by the different implementations.

The performance of middleware can be essential for determining if a specific real-time application can be migrated to the cloud. This requires to analyse the timely behavior of the middleware implementation and extract conclusions about the suitability for specific physical deployments (i.e., software, hardware, and network structure) and application types (e.g. data intensive, sporadic short messages, etc.). Also, traditional virtualization techniques can be a source of overhead and even nondeterminism. Virtualization technology comes at the cost of, in general, being more prone to suffering variations in performance compared to bare machine execution, in general. The

latter needs to be studied for the specific deployments since technological developments, such as multicore systems, are introducing new interesting properties derived from execution on dedicated cores. In a previous work [3] [2], we have performed an exploratory analysis of the performance evaluation on virtualized environments extracting preliminary results. In this paper, we deepen into the analysis of DDS in a virtualized deployment, providing a benchmark for the analysis, conducting further experiments, and elaborating conclusions as comparison between the two most popular implementations. We explore the overhead of virtualization in distributed DDS communication stacks by black box benchmarking (with no code fine-tuning), and we reason about the causes of virtualization costs, communication latencies, communication jitter, and execution nondeterminism.

The paper is structured as follows. Section 2 describes related work. Section 3 presents the potential drawbacks of the virtualization technology for timely behavior, and it describes the benchmark elaborated for the experiments or specific tests that have been carried out. In section 4, the proposed virtual data-distribution scenario is defined (two main DDS implementations running on VirtualBox) as well as the used evaluation forms, i.e. processor and network intensive scenarios. Section 5 reports the evaluation results discussing the minimum, maximum, and average response-times in different setups. Finally, section 6 outlines the main conclusions and future work.

II. Related work

Virtualization technology for cloud computing, such as hypervisors and/or virtual machine monitors, can challenge the temporal properties of soft real-time applications due to the possible introduction of higher latencies and communication jitter. Still, the deadlines for the soft real-time domain may be respected (or tolerably lost) by the new high performance cloud computing platforms that provide very efficient networking by using specific technology as InfiniBand [4].

Predictable hypervisors exist that achieve temporal and spatial isolation such as the academic initiatives of [26] [21], among others in the industrial domain[1], for real-time domains. In the hard real-time domain, predictability offered by real-time hypervisors is obtained at the cost of having to recompile the execution environment. This is not desired for the case of soft real-time applications and mainstream domains that are likely to be interested in using existing binaries, and they may even suffer run-time migration.

There are a few studies and analysis of the performance of both, virtualization technology and virtualized environments with varying quality results. Diverse applications have been used as payload to evaluate virtualization performance. These can refer to low-level services [10], function-specific applications (e.g. MapReduce [19] [16], storage solutions [20]), and middleware systems [18]. Some works report [17] significant delays due to the virtualization layer in contexts where applications are in execution within virtual machines. In contrast, other *empty* scenarios (i.e., without applications or virtual machines) report that the execution is similar to the results obtained on the physical platform [24] [25].

For this purpose, other virtualization technologies exist that do not offer temporal isolation but statistical guarantees with the advantage of allowing functional additions at run-time.[2]

The different implementations of DDS were not originally designed for virtualized environments. As a result, they can exhibit a significant different behavior either in a virtualized or in a bare machine with operating system. There are some previous experiences of using DDS in a virtual context offering good average communication times, such as the one reported in the iLAND reference implementation [6] [15] that uses a bi-dimensional QoS model [14] that can be mapped to DDS QoS properties. Possible sources of this behavior are the efficient resource management policies at node level inspired on [12] using QoS resource brokers such as [9]; timeliness was preserved even in the event of system reconfigurations that required real-time service composition [13] [32]. However, no benchmarking was performed in this context and only average times were reported.

Mainstream and traditional individual parallel applications or benchmarks have been applied to evaluating the performance of virtual machines. Benchmarks are being modified to adequately model the operation of virtual machines such as the industry benchmarks VMark [11], vConsolidate [10], and SPEC committee [22] that are virtualization benchmarks that can be used for consistent and repeatable server performance analysis. There are interesting studies applying vConsolidate in specific VM performance modeling such as [27]. Released two weeks prior to the submission of this work, [22] simulates a world-wide company with an IT infrastructure with varied requests that enables specifying deadlines for service requests (from few to hundreds of ms, and supports multiple run configuration for analysing bottlenecks at multiple layers (from hardware to application layer).

The execution of communication middleware in a virtual environment is not supported by a specific benchmark. Consequently, we have identified a set of specific tests for devising the behavior of the system to identify possible bottlenecks, reasoning about the possible sources of the problems.

III. Benchmarking virtualized middleware

The behavior of the system is analysed in terms of *usage of physical resources*, *stability* of the execution, and *load* of the servers is considered as an initial step to analyse the system. Considered resources are: Processor, network bandwidth, and memory consumption. The stability is measured by analysing the behavior of specific communications in the presence of interference and without interference. Different load levels for the servers are also experimented by executing operations that require various resource usage levels, from light weight to

[1] WindRiver Hypervisor, WMWare ESX, etc.

[2] Popular virtualization technologies that provide applications execution environments include Citrix Xen, VMWare, KVM [19], Oracle VirtualBox, SPLPAR, MS Virtual Server and Solaris Container [23].

heavy operations. Other interesting measures are derived such as *throughput* (i.e., number of requests per unit of time), and *latencies*.

A. Potential performance drawbacks in the virtualization

The execution risks of a virtualized communication middleware are the following:

- Overhead of the virtualization. Virtual machines are interfered by the execution of other VMs. This may affect the use of *visible* shared resources (e.g. the same physical core or memory capacity) and *invisible* shared resources (e.g., cache space, memory bandwidth, etc.). These can be visible or invisible depending on the implementation of the host operating system and virtualization monitor.
- Overhead of the communication middleware abstractions. A virtualization infrastructure adds extra costs in the response time of distributed applications since requests traverse the software layers; requests may be queued at different levels. This overhead affects main statistical metrics (i.e. minimum, average, and maximum response times), increasing jitter and overhead. That refers to the cost of serializing and deserializing parameters sent in different communications. Notice that part of this serialization cost may be alleviated using virtual machines that run similar virtualized operating systems and hardware infrastructures.
- Coexistence issues. Other particular inefficiencies stemmed from the integration of two different software stacks: the virtualization software and complex middleware. Depending on the particular middleware-virtualizer combination, different inefficiencies may appear (e.g. unnecessary copies from virtualized buffers to middleware buffers).

B. Benchmark description

In order to produce a meaningful set of tests for virtualized middleware, a benchmark should take into account the following key aspects:

- Application nature. Different types of applications exhibit distinct performance patterns that are, mainly, of two types: (i) *network intensive* applications and (ii) *CPU intensive* applications. Network-intensive applications make heavy use of I/O operations and peripheral actions, and their processor computations are minimum as compared to the network I/O activity. CPU intensive are dedicated to intra-node activity rather than in communication or information exchange.
- Middleware communication paradigm. The supported interaction paradigms of the middleware (e.g. its publish-subscribe (P/S), synchronous remote invocations, etc.) influence its internal implementation and synchronization aspects which directly affects the performance of the remote execution and, as a result, also influence virtualized environments. Other influencing aspects to be taken into account are the marshalling (and unmarshalling) techniques which typically represent a considerable source of overhead in middleware infrastructures.
- Virtualization software characteristics. The type of virtual machine monitor (VMM) or hypervisor and the virtualization technique, and guarantees (either real-time or statistical) over the temporal and spatial isolation of virtual machines influence the performance of the system.

Next section illustrates a practical evaluation via a specific set of tests that consider the above mentioned concepts in a general scenario: i) a client-server application, which is ii) running on DDS, which iii) is virtualized using VirtualBox over Linux. This soft real-time scenario has been chosen because it reduces development and deployment costs (i.e. the time require to develop a virtualized application). Real-time virtualizers would produce better performance results, requiring additional resources (CPU, or additional infrastructure) too.

IV. Analysis of DDS executing in virtual machines

This section describes the set of tests carried out in a client-server application installed on a DDS infrastructure. Such applications are typical of many distributed systems and require the server to block, waiting for a response from the client. In essence, the benchmarked application carries out the following operations:

- The client sends information packed in an array that is transferred to a server node. Internally, the communication with the server is carried out using a DDS topic.
- Then, the server which is another node running DDS, reads the data, processes the data, and sends back a response to the client node. In the specific implementation of the test, this action is supported with a different DDS topic that sends data back to the client.
- After receiving the information, the client to server communications stops so the client-server interaction ends.

A. Experimental setting

The physical deployment comprises two machines, one acting as a server and another as a client (see Table I). Both machines are connected via a local isolated Switched Ethernet network that connects to Linux nodes. Client and server run in a Ubuntu Linux 12.04 virtualized (with Virtualbox) image that communicates via one of two alternative DDS implementations: The first is the OpenSplice 5.5 DDS, and the second is the professional RTI 5.0 implementation.

Since the hosting operating system, the virtualization software and the virtualized operating systems are non real-time infrastructures, the tests carried out focus on average performance that may be suitable in some best-effort real-time applications. A worst-case scenario requires to use a real-time virtualizer and real-time operating system, which are not the focus of this evaluation scenario.

In this particular, the following evaluation goals were pursued:

TABLE I
HARDWARE AND SOFTWARE STACK USED IN THE EVALUATION

HW/SW Item	Description
Server machine: CPU Server machine: Memory	Core2Duo E4500 @2.2 Ghz 6 Gigabyes
Client machine: CPU Client machine: memory	Core2 6320 @1.86 Ghz 3 Gigabyes
Network	100 Mbps switched Ethernet
Hosting OS Virtualization software Hosted OS	Ubuntu 12.04 Virtualbox 4.2 Ubuntu 12.04
First DDS middleware: Second DDS middleware	OSPL Community v5.5.1 RTI Connext Professional 5.0
Small size data sets: Medium size data sets:	64 bytes 512 bytes
Processing time at server:	From 0 to 100 μs

- To measure the absolute performance of client-server applications from different DDS middleware vendors.
- To evaluate the overhead introduced by the virtualization infrastructure in different DDS implementations. To assess the differences in costs introduced by the virtualization process.
- To evaluate the impact of different virtualized DDS middleware implementations from the point of view of a real-time application (considering different deadlines).
- To determine the absolute overhead introduced by the DDS infrastructure when compared against an ideal infrastructure. The ideal infrastructure refers to a minimum distributed system based on ICMP messages that do not pay serialization/deserialization costs.

B. Results and analysis

The first experiment refers to the time required for the whole client-server interaction under different setups. The different setups refer to the following choices:

- The experiment is executing (i) inside the virtual machine or (ii) in the host with no VM intermediation.
- The experiment is running (i) on an ideal ICMP scenario, (ii) on OSPL or on (iii) RTI stacks.
- In the experiment the data sent to the server has to be processed. The processing at the server ranges from 0 to 100 μs.

The obtained results (see Figure 1) show the expected performance patterns. In all cases, the execution costs increase with the amount of data sent to the server. They also increase as they are virtualized, i.e. the costs in the non virtualized environment are less than in the virtualized one, ranging from 800μs to few milliseconds with medium size data sets.

A remarkable result is the gap between the ideal middleware setting (represented in the evaluation with ICMP) and DDS. It is due to the multiple abstractions that are supported by the DDS programming model, mainly due to serialization overhead, and to the use of topics and multiple I/O buffers, that manifest (i.e., are paid for) in the ICMP stack.

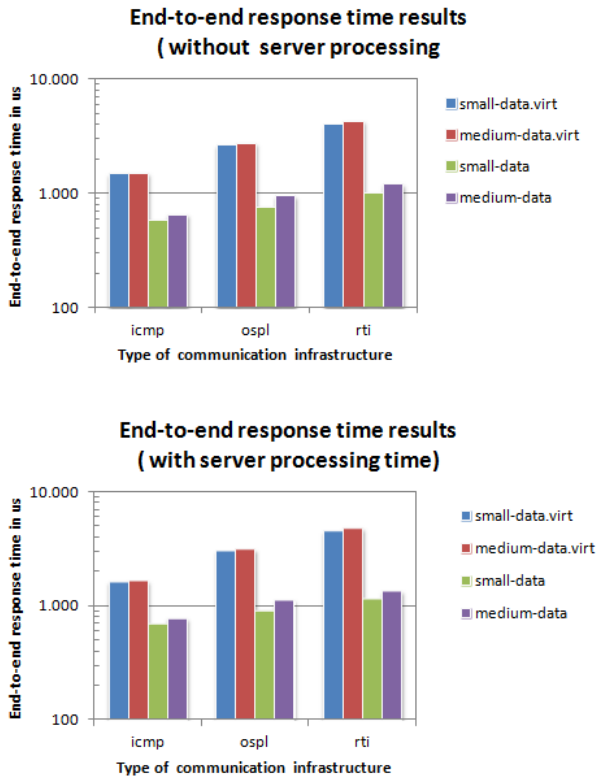


Fig. 1. Absolute end-to-end response time results with server and without server processing time

Figure 2 complements the previous results with information on the extra cost paid by the virtualization process. For the given scenarios, the extra cost ranges from a minimum of 120% to a maximum of almost 300%. In practical terms, the virtualized application has reductions in performance that may leave the available utilization in almost 25% of the time consumed in a non virtualized environment equivalent. Notice that this time is, to some extent, the maximum penalty; this could be alleviated by using optimized virtualizers that take into account the host infrastructure. The virtualizer used in this experiment does not take advantage of this feature to improve performance.

It is also remarkable that the virtualization may require up to 50% of the total available time for small response time applications (i.e., applications with a $10ms$ deadline). This cost is reduced to less than 5% (i.e. a more moderated and admissible penalty) when deadlines are in the $100ms$ range. As operational deadlines increase, this margin reduces to 1% for applications with deadlines that are in the range of milliseconds.

The last set of experiments refers to the overhead introduced by a middleware like DDS. Different middleware implementations introduce an overhead when they compare against an idealized communication middleware that do not require to perform general application serialization, copying data from different multilevel buffer, nor other middleware-

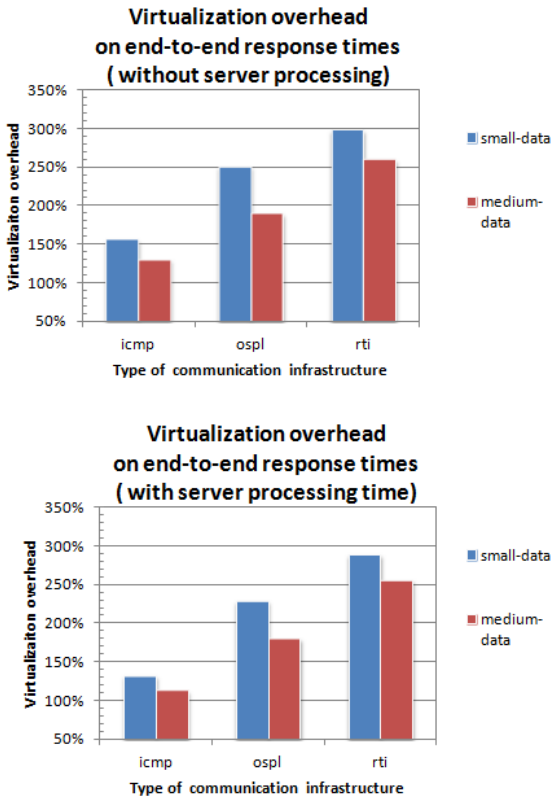


Fig. 2. Virtualization overhead main results with and without server processing time

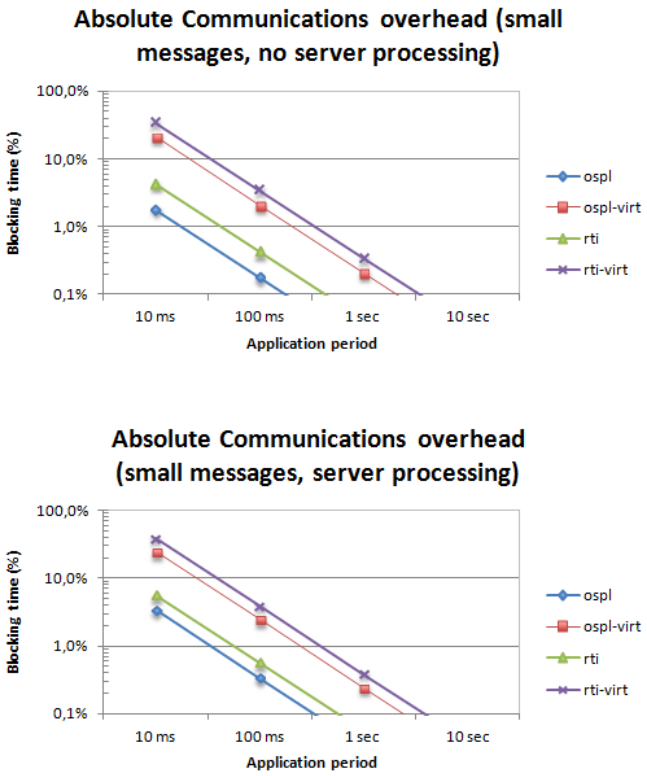


Fig. 3. Overhead introduced by virtualized middleware technology (small size data)

level overhead. As in the previous cases, the evaluation has been carried out in small (see Figure 3) and medium (see Figure 4) data sizes.

The following are remarkable outcomes:

- In most cases, the overhead introduced by the stacks represents an important amount of the available time. This extra overhead takes into account the amount of time required for serialization and deserialization processes.
- For the given virtualization scenarios (and under the described evaluation conditions), the use of OSPL support outperforms an RTI equivalent stack. In average performance terms, the virtualized RTI requires and 50% amount of CPU time to offer an OSPL-equivalent performance.
- Lastly, it should be noticed that for both implementations, the overhead of the virtualization dominates over the overhead introduced by the middleware abstraction. In all tests (see Figure 3 and Figure 4), the cost of the middleware abstraction is typically 30% of the total time, while the the cost of the virtualization may represent 72% of the total time. In practice, this effect is shown in the graphs with the two virtualized DDS-middlewares as virtualized implementations consume more resources than their non virtualized equivalents.

V. Conclusion

The work describes a benchmarking process to obtain information on the performance of virtual machines containing applications that communicate via publish-subscribe (data centric) middleware. Precisely, we have analysed the behavior of DDS for its two most popular implementations (Open Splice and RTI). Initially, we have identified the important aspects to consider in the design of a benchmark for performance analysis of virtualized middleware, including the identification of the potential bottlenecks to search for, and the considerations with respect to the software stack to be analysed. Lastly, we have describe the benchmark tests executed for applications that make intensive use of the network and the processor. Results have shown the comparison and impact on both implementations of the virtualization software.

Future work will include the execution of just released industrial benchmarks for virtual machines that simulate a real environment based on scenarios described in [28], [29] and [30].

Acknowledgment

This work has been partly supported by the Salvador de Madariaga Programme for International Research Stays from the Spanish Ministry of Education (PRX12/00252) and by the Spanish national project REM4VSS (TIN 2011-28339).

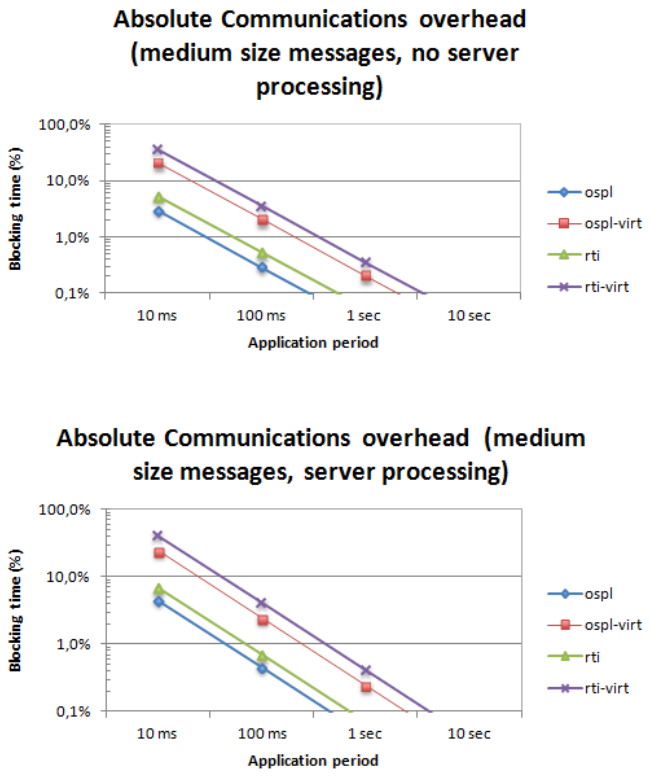


Fig. 4. Overhead introduced by virtualized middleware technology (medium size data)

REFERENCES

[1] Object Management Group. *Data Distribution Service for Real-Time Systems.* OMG, 1.2 formal/07-01-01 edition., Jan. 2007.

[2] R. Serrano-Torres, M. García-Valls, P. Basanta-Val. *Performance Evaluation of Virtualized DDS Middleware* IV Simposio de Sistemas de Tiempo Real (IV STR - CEDI). September 2013.

[3] R. Serrano-Torres, M. García-Valls, P. Basanta-Val. *Virtualizing DDS middleware: performance challenges and measurements*. IEEE International Conference on Industrial Informatics (INDIN'13). Bochum, Germany. July 2013.

[4] Infiniband Trade Association. *InfiniBand Architecture specification vol. 1, release 1.2.1.* 2007.

[5] G. Chen, M. Li, and D. Kotz. *Data-centric middleware for context- aware pervasive computing* Pervasive and Mobile Computing, pp. 216 253, April 2008.

[6] M. García-Valls, L. Fernández Villar, I. Rodríguez López. *iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time Systems*. IEEE Transactions on Industrial Informatics, vol. 9(1), pp. 228- 236. February 2013.

[7] iLAND project. *iLAND Reference Implementation Installation & User Guide.* September 2012. `http://sourceforge.net/projects/iland-project/`

[8] M. García-Valls, A. Crespo, J. Vila. *Resource Management for Mobile Operating Systems based on the Active Object Model* International Journal on Computer Systems Science and Engineering. July 2013.

[9] M. García-Valls A. Alonso, J. Ruíz, A. Groba. *An architecture for a Quality of Service resource manager for flexible multimedia embedded systems*. In Proc. of 3rd International Workshop on Software Engineering and Middleware (SEM02). Lecture Notes in Computer Science vol. 2596. 2003.

[10] M. Greenfield, J.P. Casazza, and K. Shi. *Redefining server performance characterization for virtualization benchmarking* August 2006.

[11] VMWare. *VMWare vmark*. http://www.wmware.com/products/vmmark/results.html

[12] M. García-Valls, A. Alonso, J. A. de la Puente. *A dual priority assignment mechanism for dynamic QoS resource management*. Future Generation Computer Systems, vol. 28(6), pp.902-911. June 2012.

[13] M. García-Valls, P. Basanta-Val. *A real-time perspective of service composition: key concepts and some contributions*. http://dx.doi.org/10.1016/j.sysarc.2013.06.008 Journal of Systems Architecture, Elsevier. (Available online 16 July 2013)

[14] M. García-Valls, P. Basanta-Val, M. Marcos, E. Estevez. *A bi-dimensional QoS model for SOA and real-time middleware* International Journal of Computer System Science and Engineering, vol. 28. ISSN 0267 6192. September 2013.

[15] M. García-Valls, P. Basanta-Val. *Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems*. http://dx.doi.org/10.1016/j.sysarc.2013.08.010 Journal of Systems Architecture, Elsevier. (Available online 24 August 2013)

[16] D. d. Oliveira, K. Ocaña, E. Ogasawara, J. Dias, J. Goncalves, F. Baiao, and M. Mattoso. *Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows* Future Generation Computer Systems, January 2013.

[17] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. *Performance evaluation of virtualization technologies for server consolidation*. HP Laboratories Palo Alto, Enterprise Systems and Software Laboratory, Tech. Rep., 2007.

[18] Oracle. *Middleware virtualization* http://www.oracle.com/us/\-technologies/virtualization/middleware-virtualization-068082.html

[19] A. Matsunaga, M. Tsugawa, and J. Fortes. *Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications*. 4^{th} IEEE International Conference on eScience, 2008. [Online]. Available: http://graal.ens- lyon.fr/ ecaron/m2/papers/papier06.pdf

[20] D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, and T. P. Lee. *Performance virtualization for large-scale storage systems*. 22^{nd} Int'l Symposium on Reliable Distributed Systems. 2003.

[21] S. Xi, J. Wilson, C. Lu, and C. Gill. *RT-Xen: Towards real-time hypervisor scheduling in Xen*. Washington University in St. Louis, Tech. Rep. http://www.cse.wustl.edu/lu/papers/emsoft11.pdf. 2011.

[22] Standard Performance Evaluation Corporation. *Specjbb2013* http://www.spec.org/jbb2013/ 2013.

[23] E. Benjamin. *3 key trends in middleware virtualization*. VMWare, vFabric Team, Tech. Rep. http://blogs.vmware.com/vfabric\-/2012/08/3-key-trends-in-middleware-virtualization.html 2012.

[24] J. Lu, L. Makhlis, and J. Chen. *Measuring and modeling the performance of the Xen VMM*. bMC Software Inc.

[25] J. Lei, X. Yang, G.Xiong, W. Jiang, and Y.Liao. *VMM-based real-time embedded system* International Conference on Embedded Software and Systems Symposia, pp. 213218. July 2008.

[26] A. Crespo, I. Ripoll, and M. Masmano. *Partitioned embedded architecture based on hypervisor: The XtratuM approach*. European Dependable Computing Conference, pp. 67-72. 2010.

[27] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell. *Modeling virtual machine performance: challenges and approaches.* SIGMETRICS Perform. Eval. Rev. vol.37(3), pp. 55-60. January 2010.

[28] P. Basanta Val, M. Garcia-Valls, M. Baza-Cuado. *A simple data-muling procotol*. Accepted in, IEEE Transactions on Industrial Informatics. 2013.

[29] P. Basanta Val, M. Garcia-Valls. *A Distributed Real-Time Java-centric Architecture for Industrial Systems*. Accepted in, IEEE Transactions on Industrial Informatics. DOI: 10.1109/TII.2013.2246172. 2013.

[30] P. Basanta Val, M. Garcia-Valls. *Resource Management Policies for Real-time Java Remote Invocations*. Accepted in, Journal of Parallel and Distributed Computing. DOI: 10.1016/j.jpdc.2013.08.001. 2013.

[31] P. Basanta-Val, M. Garca-Valls. *Enhanced JRMP multiplexing headers under non-fragmented local area network constraints*. Electronics Letters, vol.49, no.21, pp.1333-1335. October 2013. doi:10.1049/el.2013.2239

[32] M. García-Valls, P. Basanta-Val. *Low complexity reconfiguration for real-time data-intensive service-oriented applications*. Future Generation Computer Systems, Elsevier. (Accepted for publication: October 2013)

A Feasible Configuration of AFDX Networks for Real-Time Flows in Avionics Systems

Dongha An, Hyun Wook Jeon, Kyong Hoon Kim, and Ki-Il Kim
Department of Informatics
Gyeongsang National University
Jinju-daero 501, Jinju, 660-701, South Korea
Email: {dhan,krsoftno1,khkim,kikim}@gnu.ac.kr

***Abstract*—AFDX (Avionics Full Duplex Switched Ethernet) Networks have been proposed to meet unique ADN (Aircraft Data Networks) characteristics and then standardized as a Part 7 in ARNIC 664. As for this new communication technology, some research works have been conducted to address design issues such as optimizing virtual links as well as analytic modeling including response time. Despite of their research efforts, configuration problem for both MTU (Maximum Transmission Unit) and BAG (Bandwidth Allocation Gap) over virtual links in AFDX networks remains unsolved yet. In this paper, we propose how to set MTU and BAG value on each virtual link according to both application requirements and AFDX switch constraints. We define a new problem of feasible configurations of virtual links in an AFDX switch and propose an algorithm to derive feasible BAG and MTU pairs based on the branch-and-bound technique. Throughout simulations, we evaluate the proposed algorithm and analyze the effect of parameters in AFDX networks.**

I. Introduction

As new aircraft's demanding requirements to high available bandwidth, minimum wiring to reduce the weight and low development cost have emerged, the current three main ADNs (Aircraft Data Networks), ARNIC 429, MIL-STD-1553 and ARNIC 629 are regarded as not appropriate communication technologies to meet these demands completely. This fact implies that not only reliable and deterministic property of ADN but also implementation cost should be concerned in next generation aircraft. Consequently, from development of data networks on the Airbus 380 aircraft, a new technology, called AFDX (Avionics Full Duplex Switched Ethernet), has been implemented and then standardized for new ADN [1], [2], [3].

The AFDX was extended from original Ethernet to ensure deterministic behavior and high reliability in order to comply with the stringent requirements of ADNs. To ensure them, new functions are implemented in two ways. One is traffic control by guaranteeing the bandwidth of each application, and the other is dual redundant channel for reliability. While the former targets to limit the jitter and transmit latency, the latter transmits the same data stream over disjoint networks. To achieve this goal, virtual links have been employed between source and destination. With these virtual links, deterministic behaviors are guaranteed and all controls are ensured through them. So, determining virtual link properties and configuring network environments become network designer's great task.

System configuration parameters of virtual links include traffic scheduling, maximum jitter, and bandwidth constraints [1], [2], [3]. Among many system parameters, two are important with regard to the guarantee of real-time requirements: *BAG* (Bandwidth Allocation Gap) and *MTU* (Maximum Transfer Unit). BAG is a timeslot confining the virtual link's bandwidth by defining the minimum gap time between two consecutive frames. The range of the BAG value is between 1 and 128 msec in a form of power of 2. MTU is defined as the maximum size of message to be transmitted in each frame.

Much recent work has focused on the system analysis of AFDX networks [7], [8], [9], [10], [12]. The AFDX network analysis is done by queuing networks, network calculus, or model checking. Throughout the analysis, the impact of parameters has been analyzed, including end-to-end delays, worst-case latencies, and so on. However, only a few studies have been done on the problem of AFDX configuration such as BAG and MTU. In [4], the authors proposed how to set the transmission parameters of virtual links so as to minimize the reserved bandwidth while transmitting the data within their maximum delivery times. They first derive optimized parameters of each virtual link for a given set of messages. Then, they solve the optimization problem of multiple virtual links in order to minimize bandwidth. The weakness of this approach is that the optimized parameters found in a single virtual link cannot be feasible when they are used in finding feasible configurations of multiple virtual links in an AFDX network switch.

In this paper, we focus on finding feasible BAG and MTU parameters of virtual links in an AFDX switch for a given virtual links of messages. We define a new problem of feasible configuration of an AFDX switch, and then solve the problem using the branch-and-bound technique. The main contributions of this paper include (i) defining a problem of feasible configuration, (ii) providing an algorithm to solve the problem, and (iii) analyzing the algorithm through the simulations.

The rest of this paper is organized as follows. First, we describe the related work briefly in Section II. And then, the system model and the problem definition are provided in Section III. In Section IV, we explain the proposed algorithm. Performance evaluations are shown in Section V. Finally, conclusion and further work are followed in Section VI.

II. Related Work

In this section, we briefly introduce related work on AFDX networks. In this research area, existing technologies mainly fall into two main categories. One is for design issue and the other is for analysis modeling.

First, the authors in [4] have focused on the design of virtual links in AFDX networks. In their works, the problem domain is ranged from how to set the transmission parameters of virtual links to how to route virtual links in the AFDX interconnect. For this goal, several closed-form results and efficient numerical algorithms as well as exact integer-linear programming formulation of the routing problem are newly presented. Through above method, optimal bandwidth management is achieved, such as, minimizing reserved bandwidth and the bandwidth consumption. In another research work, modeling method for AFDX frame management was introduced to ascertain the reliability properties of design [5]. They modeled the system as a network of timed automata to indicate weakness of current AFDX frame management against faults. Moreover, they present the solution by including a priority queue at receivers. In addition to mentioned works, one of outstanding features, reliability through redundancy transmission on AFDX was analyzed by formal method in [6].

While the design issue targets to build AFDX networks, the other works have been proposed to analyze the system metric such as response time. The representative work for this goal has been proposed in [7]. The authors introduce three methods, network calculus, queuing networks simulation and model checking to evaluate bounding end-to-end delays on AFDX networks. As the previous work, they also showed that Trajectory approach which analyzes the worst-case delays throughout message flows outperforms the Network calculus method under industrial configuration [11] and reached reliable conclusion that combination of two methods could lead to an improvement of the existing analysis in [8]. However, since the previous model did not include contention in the end or switches, different analysis was given to obtain worst-case latencies and output jitter for the network messages in [9] by defining a real-time model for a communications network based on AFDX. In addition to analysis model, simulation system based on popular NS-2 was designed and implemented to evaluate the performance and analyze impact of several system parameters such as scheduling algorithm in [10].

III. System Model and Problem Definition

A. System Model

Avionics network systems consist of many components, such as sensors, LRUs (Line Replacement Units), computing units, and so on. These components communicate each other throughout AFDX switches. An AFDX message is uniquely defined by UDP source and destination ports, as shown in Figure 1. Sine we focus on real-time AFDX messages, a message flow f_i is defined by (l_i, p_i), where l_i is the payload of the message in bytes and p_i is Message Transmit Cycle (MTC) of the message in msec. That is, a message of l_i bytes is generated every p_i time units and is delivered to the destination application.

A *Virtual Link (VL)* is a logical communication unit in AFDX networks. Figure 1 shows an example of AFDX

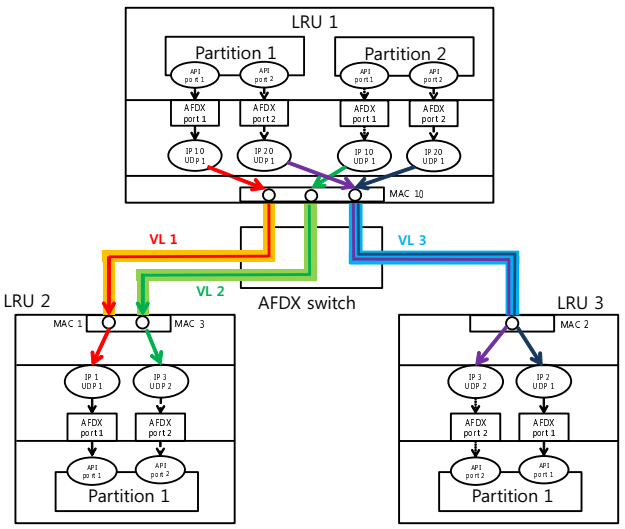


Fig. 1. An example of virtual links in an AFDX switch

networks with three virtual links among LRUs. These virtual links sharing physical links are scheduled in AFDX network switches. Furthermore, multiple applications transmit real-time messages throughout a common virtual link if their source and destination units are the same. In the example of Figure 1, two application messages are shared in the virtual link VL_3.

A virtual link requires two important parameters other than source and destination information. The first is Bandwidth Allocation Gap (BAG) to specify a periodic frame. In AFDX switches, a BAG is defined by a value of 2^k msec, where $k = 0, 1, \ldots, 7$. As all BAGs are 2^k msec, virtual links are multiplexed in AFDX switches. The second parameter is Maximum Transfer Unit (MTU) of the message in bytes at each frame. Payloads of applications in a virtual link are transmitted within maximum MTU bytes in a single frame. If the size of a payload is greater than the MTU, it is fragmented into multiple frames. Therefore, a virtual link VL_i is defined by (BAG_i, MTU_i, F_i) as follows.

- BAG_i: bandwidth allocation gap or period of VL_i in a value of 2^k msec where $k = 0, 1, \ldots, 7$.
- MTU_i: maximum transfer unit or message size of VL_i in bytes.
- F_i: a set of message flows in VL_i, where the j-th message flow is denoted as $f_{i,j} = (l_{i,j}, p_{i,j})$.

As avionics systems are hard real-time systems, it is an important issue to guarantee the schedulability both in computing units and in network flows. The virtual link scheduler in an AFDX switch plays a role in scheduling multiple virtual links. For example, the scheduling algorithm in [2] is Round Robin (RR). In this paper, we will define a new problem of finding a feasible configuration of BAG and MTU pairs of given virtual links in an AFDX switch in order to meet all real-time requirements of messages.

B. Problem Definition

For a given virtual link VL_i, MTU and BAG are configured so as to meet all the real-time requirements of message flows in

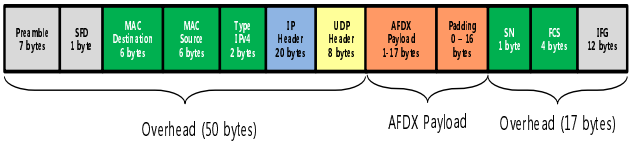


Fig. 2. The AFDX frame structure and its overhead

the link. If the payload of a message is greater than the MTU size, it is transmitted in multiple fragmented packets. Since all BAGs of VLs are harmonic, the schedulability analysis is easily derived by utilization analysis. Thus, Eqn. (1) tells the message constraint of VL_i with n_i messages to guarantee the real-time requirement of all message flows in the link [2].

$$\sum_{j=1}^{n_i} \frac{\lceil l_{i,j}/MTU_i \rceil}{p_{i,j}} \leq \frac{1}{BAG_i} \tag{1}$$

Let us assume that the system has N VLs on an AFDX switch with B bandwidth in bps. Each VL_i is configured with (MTU_i, BAG_i), so that MTU_i bytes are transmitted every BAG_i msec. In addition, each VL message requires the overhead of 67 bytes as shown in Figure 2. Since the total bandwidth of VLs should not exceed the network bandwidth, the following bandwidth constraint should be met.

$$8 \sum_{i=1}^{n} \frac{MTU_i + 67}{BAG_i} \times 10^3 \leq B \tag{2}$$

The last constraint of virtual link scheduling is about jitter. The maximum allowed jitter on each virtual link in the ARINC 664 specification requires 500 μsec [2]. Thus, the following equation tells the jitter constraint, where 40 μsec is the typical technological jitter in hardware level to transmit an Ethernet frame.

$$40 + \frac{8 \sum_{i=1}^{n} (67 + MTU_i)}{B} \leq 500 \tag{3}$$

Now we define a problem of finding a feasible configuration of BAG and MTU pairs of virtual links of an AFDX switch. Three constraints of Eqn. (1), Eqn. (2), and Eqn. (3) should be met in order to satisfy all real-time requirements of messages in virtual links, which derives a new problem as follows.

Definition 3.1: For a given set of virtual links $V = \{VL_i \mid i = 1, \ldots, N\}$, the problem of **AFDX-CONF** is to determine (BAG_i, MTU_i) of each VL_i so as to satisfy three constraints of Eqn. (1), Eqn. (2), and Eqn. (3), where $BAG_i \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ and $MTU_i \in \{1, 2, \ldots, 1471\}$.

IV. The Proposed Algorithm

We solve the problem **AFDX-CONF** in two steps. The first step is to find the list of (BAG_i, MTU_i) which guarantees the schedulability of message flows in VL_i. Each (BAG_i, MTU_i) should be selected such that it satisfies the constraint of Eqn. (1). Then, we find the feasible solutions of a given virtual links with consideration of two constraints of Eqn. (2) and Eqn. (3).

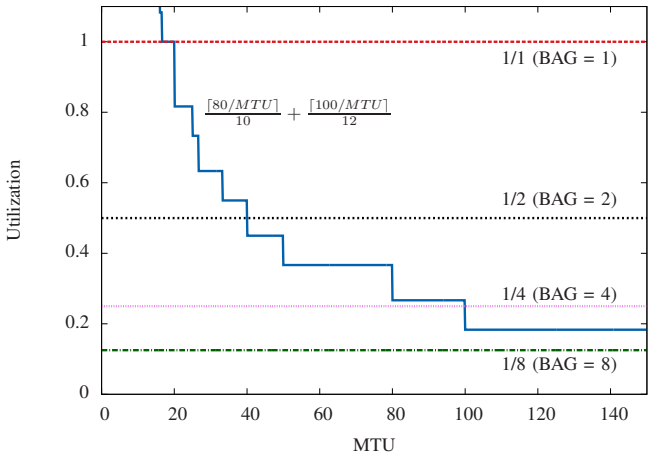


Fig. 3. An example of feasible BAG and MTU of a virtual link

A. Schedulable BAG and MTU Pairs of a VL

Let us consider a virtual link VL_1 with two message flows of $f_{1,1}(80, 10)$, $f_{1,2}(100, 12)$ as an example. The values of BAG and MTU of VL_1 are set to satisfy Eqn. (1) in order to meet the real-time requirement of two messages. The left side of Eqn. (1) is shown in Figure 3 as a step function, while $1/BAG$ is also drawn in the figure for different BAG values.

For a given BAG_i, there exist many MTUs which satisfy Eqn. (1). For example, when BAG_1 = 1, all MTUs can be used if MTU $\geq$ 17, as shown in Figure 3. Since a longer MTU size requires more bandwidth and jitter, the smallest value should be selected. Thus, MTU_1 of the example VL_1 is 17 bytes when BAG_1 is 1 msec. Similarly, MTUs of VL_1 for BAGs with 2 msec and 4 msec are given by 40 bytes and 100 bytes in each, as shown in Figure 3.

When the MTU size is greater than the maximum payload size of messages, the required utilization is not changed. For example, the lower bound of the utilization of VL_1 is given by about 0.1834 at MTU = 100. This implies that there is no MTU which guarantees the schedulability of two messages if BAG is greater than or equal to 8 msec. Therefore, the feasible solutions, (BAG_1, MTU_1), of VL_1 are given by (1, 17), (2, 40), and (4, 100).

The pseudo-algorithm of Figure 4 describes how to obtain the set of feasible BAG and MTU pairs of a given virtual link VL_i. The first part of the algorithm gathers all step integers at which the utilization function begins a new piecewise constant due to the ceiling function. We denote the set of such step integers as N_{step}. For each message $f_{i,j}$, such step points are derived and added into N_{step} (lines 1-8).

Then, for each 2^k value, we find the minimum MTU which satisfies Eqn. (1). (lines 9-13). We denote $s_{i,k}$ as the feasible BAG and MTU pair in case of $BAG_i = 2^k$ for a virtual link VL_i. For a given n_i flows, the time complexity of the algorithm in Figure 4 is $O(n_i \cdot |N_{step}|)$ since we have to find and check the feasibility at each step point of messages.

Algorithm Find_Feasible_BAG_MUT (VL_i)

$N_{step} \leftarrow \varnothing$
for each message $f_{i,j}$ in VL_i **do**
  frag $\leftarrow \lceil l_{i,j}/(\lceil l_{i,j}/p_{i,j} \rceil) \rceil$
  while *frag* ≥ 1 **do**
    $m \leftarrow \lceil l_{i,j}/frag \rceil$
    $N_{step} \leftarrow N_{step} \cup \{m\}$
    frag $\leftarrow$ *frag* $- 1$
  endwhile
endfor
for k **from** 0 **to** 7 **do**
  $m_k \leftarrow$ the least $m \in N_{step}$ s.t. $\sum_{j=1}^{n_i} \frac{\lceil l_{i,j}/m \rceil}{p_{i,j}} \leq \frac{1}{2^k}$
  if $m_k \neq$ *NULL* **then**
    $s_{i,k} \leftarrow (2^k, m_k)$
endfor

Fig. 4. Algorithm of feasible BAG and MTU pairs of a VL

B. Feasible BAG and MTU Pairs of VLs

The problem of finding feasible BAG and MTU pairs of a given set of virtual links is not trivial. For example, let us consider the example of two virtual links of Table I where the network speed (B) is given by 1Mbps. For each virtual link, the feasible BAG and MTU pairs are derived by the algorithm of Figure 4, as shown in the last column of Table I. Now, a new problem arises about selecting appropriate BAG and MTU pairs of two virtual links so as to meet both constraints of Eqn. (2) and Eqn. (3).

There are some tradeoffs among feasible $s_{i,k}$ of a virtual link VL_i. Solutions with smaller BAG provide less jitter due to smaller MTU size, while they require more bandwidth due to overhead of fragmentation. For example, if we select (1,5) and (1,6) as (BAG, MTU) of two VLs of Table I, it does not meet the bandwidth constraint of Eqn. (2). On the contrary, if (2,9) and (2,12) are selected as (BAG, MTU) of two VLs, this configuration does not meet the jitter constraint of Eqn. (3). The selection of (1,5) and (2,12) of VL_1 and VL_2 satisfies both constraints so that all messages in VLs meet their real-time requirements.

Let us denote $s_{i,k}$ as the feasible BAG and MTU pair of VL_i in case of $BAG_i = 2^k$, which is derived from the algorithm of Figure 4. If there is no feasible MTU for $BAG_i = 2^k$, $s_{i,k} = \varnothing$. Then, the problem to be solved is defined as follows.

Definition 4.1: For a given set of virtual links $V = \{VL_i \mid i = 1, \ldots, N\}$, let us assume that a feasible pair of BAG and MTU for $BAG_i = 2^k$ is available as $s_{i,k}$. The problem of **AFBM** is to select $s_{i,k}$ of each VL_i so as to satisfy both constraints of Eqn. (2) and Eqn. (3).

For a given N virtual links, the exhaustive search of the problem **AFBM** takes $O(8^N)$ since each virtual link might have maximum eight solutions. In this paper, we provide a branch-and-bound algorithm to find a feasible solution for a given N virtual links with their feasible BAG and MTU pairs derived by Figure 4. The proposed branch-and-bound algorithm consists of pruning condition and branch-and-bound strategy as follows.

TABLE I. AN EXAMPLE OF VIRTUAL LINKS (B = 1 MBPS)

	Flows ($f_{i,j}$)	Payload ($l_{i,j}$)	MTC ($p_{i,j}$)	Feasible BAG and MTU pairs ($s_{i,k}$)
VL_1	$f_{1,1}$	200	80	(1,5), (2,9), (4,17), (8,34), (16,67), (32,200)
	$f_{1,2}$	250	160	
VL_2	$f_{2,1}$	250	220	(1,6), (2,12), (4,25), (8,50), (16,100), (32,200)
	$f_{2,2}$	200	40	

Algorithm Find_Feasible_Configurations (V)

/* $V = \{VL_i | i = 1, \ldots, N\}$ */
1: **for** i **from** 1 **to** N **do**
2: call **Find_Feasible_BAG_MUT** (VL_i)
3: $S \leftarrow \varnothing$
4: *result* $\leftarrow$ **DFS_BandB** (0, 0, 1, S)
5: **return** S

Function DFS_BandB (B_{curr}, J_{curr}, i, S)
6: **if** $i = N + 1$ **then return** *true*
7: **for each** $s_{i,k}$ of VL_i **do**
8: *bandwidth* $\leftarrow B_{curr} + (mtu_{i,k} + 67)/bag_{i,k}$
9: *jitter* $\leftarrow J_{curr} + 67 + mtu_{i,k}$
10: **if** *bandwidth* $\leq B/8000$ **and** *jitter* $\leq 460 \cdot B$ **then**
11: *result* $\leftarrow$ **DFS_BandB** (*bandwidth*, *jitter*, $i + 1$, S)
12: **if** *result* $=$ *true* **then**
13: $S \leftarrow S \cup \{s_{i,k}\}$
14: **return** *true*
15: **endif**
16: **endif**
17: **endfor**
18: **return** *false*

Fig. 5. The proposed algorithm

- **Pruning condition:** The pruning condition is two constraints of Eqn. (2) and Eqn. (3). The algorithm examines whether the solutions in the subset satisfy both constraints. Since both bandwidth and jitter values increase with a new branch in the search tree, the algorithm stops the search of the subset which already violates one of two constraints.
- **Branch and bound strategy:** We can use the current values of total bandwidth and jitter as a branch condition. For example, a node with the least bandwidth is selected as a new branch. The algorithm finds a feasible solution when it reaches at any leaf node in the search tree.

The proposed algorithm searches a feasible solution in a leaf node in Depth-First-Search (DFS) manner. The function **DFS_BandB** in Figure 5 is the recursive implementation at level i in the search tree. Two values of B_{curr} and J_{curr} are the total bandwidth and jitter of sub-solutions from VL_1 to VL_{i-1}. For each $s_{i,k} = (bag_{i,k}, mtu_{i,k})$, two constrains of Eqn. (2) and Eqn. (3) are checked including a new solution of VL_i (lines 8-10). If either of two constraints is not satisfied, it is pruned. Otherwise, the depth-first-search is continued with two updated bound values (line 11).

When the search reaches at a leaf node, the function returns *true* (line 6). The return value of calling **DFS_BandB** is *true*, the final solution S is updated as to include $s_{i,k}$ (line 13) and the function returns *true*. Thus, the problem of **AFDX-CONF** is solved by the algorithm in Figure 5. If the return value of **DFS_BandB** (0, 0, 1, S) is *true*, a feasible solution is stored

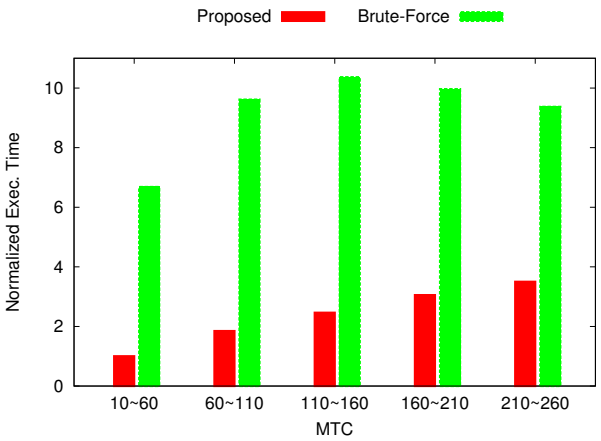


Fig. 6. Algorithm execution time

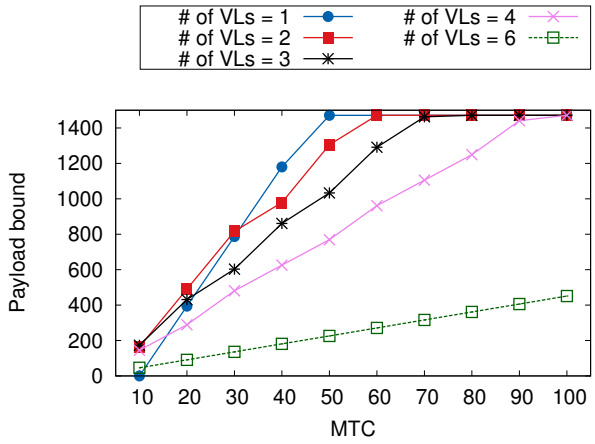


Fig. 7. Payload bounds w.r.t MTC

TABLE II. THE PERCENTILE OF FEASIBLE SOLUTIONS

MTC	$10 \sim 60$	$60 \sim 110$	$110 \sim 160$	$160 \sim 210$	$210 \sim 260$
Feasible Sets	3.4%	23.6%	40.1%	54.0%	62.7%

in S. Otherwise, the empty set is returned, which implies no feasible configuration is found for a give set of virtual links.

V. PERFORMANCE EVALUATIONS

In this section, we show performance evaluation of the proposed algorithm. First, we evaluate the execution time of the proposed algorithm compared with the brute-force search. In the experiments, we generate five virtual links with two message flows in each virtual link. The payload of a message is randomly generated from 20 to 80 bytes. The MTC or period of a message is randomly selected among five different intervals, as shown in Figure 6. The network bandwidth is set as 6Mbps.

For each case of Figure 6, we generate 5000 random sets of five virtual links and measure the average execution time of the proposed algorithm. In order to compare the execution time, the brute-force search algorithm is also run. In Figure 6, the execution time is normalized based on that of the proposed algorithm in case of the first interval of MTC in the experiments.

As shown in Figure 6, the proposed algorithm runs about two or six times faster than the exhaustive search algorithm. Since the proposed algorithm is based on branch-and-bound technique, it runs faster. Table II shows the percentile of feasible solutions among 5000 random test cases. In case of smaller MTCs, the performance of the proposed algorithm is more than those in bigger MTCs. As shown in Table II, most of random test sets are infeasible in lower MTCs. In this case, the proposed algorithm rejects the given virtual link sets in early search steps due to the pruning condition. However, the exhaustive search algorithm tests all possible cases.

Next, we analyze the payload bound of a message to be schedulable. The MTC is varied from 10 to 100 bytes. We generate 12 messages of the same requirement. The number of virtual links is varied from 1 to 6 in order to analyze the impact of the number of virtual links. Figure 7 shows the payload bound of a message by simulating the proposed algorithm from the payload size 1 to 1471. Figure 7 shows that the message of lower payload size than the bound is guaranteed to be scheduled.

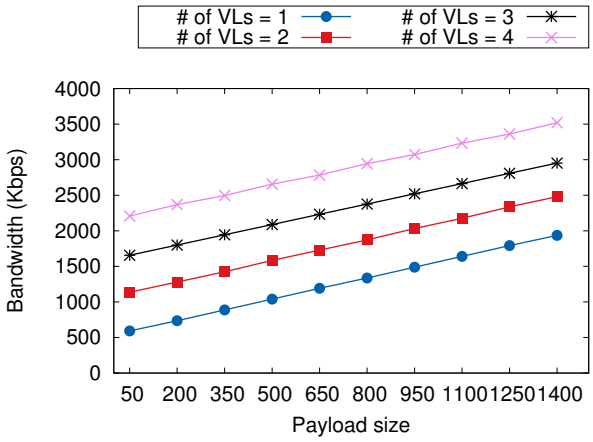
As shown in Figure 7, the schedulability of more virtual links shows generally worse than that of less virtual links for the same number of messages. It is because of jitter and bandwidth overhead of virtual links in AFDX switches. However, in case of lower MTCs, the schedulability of a single virtual link shows poor since it becomes difficult to meet the message constraint of Eqn. (1).

Let us consider the case of MTC = 100 in Figure 7. All messages of the payload size less than or equal to 1471 bytes are schedulable if $N \leq 4$. We measure the bandwidths and jitters of four different number of virtual links, as shown in Figure 8. Figure 8 implies that it is better to use a single virtual link to send 12 messages in order to reduce the total bandwidth and jitter. The remaining bandwidth can be used to transmit other non-real-time network traffic in AFDX switches.

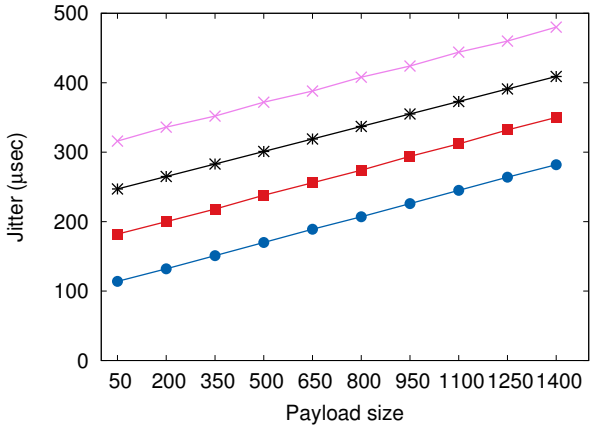
VI. CONCLUSIONS

In this paper, we defined a new problem of feasible configurations of an AFDX switch for the purpose of meeting the real-time requirements of all messages in avionics. Two important parameters of BAG and MTU of virtual links are derived by solving the problem. The proposed algorithm first derives optimal MTUs of a virtual link for each possible BAG, and then obtains feasible BAG and MTU pairs of multiple virtual links. In the simulation results, the proposed scheme is faster than the exhaustive search algorithm. And, we also analyzed the payload bound and the effect of selection of virtual links.

Since the AFDX network configuration becomes an important issue in avionics systems, we will investigate many problems based on the results of this paper. For example, we will extend the problem into multiple AFDX switches or discuss about the routing issues through the networks.



(a) bandwidth



(b) jitter

Fig. 8. Bandwidth and jitter of feasible sets

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2012R1A1A1015096) and BK21+ Program.

REFERENCES

[1] R. L. Alena, J. P. Ossenfort, K. I. Laws, A. Goforth, and F. Figueroa. Communications for integrated modular avionics. In *Proc. of 2007 IEEE Aerospace Conference*, March 2007.

[2] AFDX Tutorial. http://www.techsat.com/fileadmin/media/pdf/infokiosk/TechSAT_TUT-AFDX-EN.pdf.

[3] AFDX/ARNIC 664 Tutorial. http://www.cems.uwe.ac.uk/∼ngunton/afdx_detailed.pdf.

[4] A. Al Sheikh, O. Brun, M. Cheramy, and P.-E. Hladik. Optimal design of virtual links in AFDX networks. *Real-Time Systems*, vol. 49, no. 3, pp. 308-336, May 2013.

[5] I. Saha and S. Roy. A finite state modeling of AFDX frame management using spin. *Lecture Notes in Computer Science*, vol. 4346, pp. 227-243, 2007.

[6] J. Taubrich and R. Hanxleden. Formal specification and analysis of AFDX redundancy management algorithms. *Lecture Notes in Computer Science*, vol. 4680, pp. 436-450, 2007.

[7] H. Charara, J. Scharbarg, J. Ermont, and C. Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *Proc. Of 18th Euromicro Conference on Real-time Systems*, pp. 193-202, July 2006.

[8] H. Bauer, J. Scharbarg, and C. Fraboul. Worst-case end-to-end delay analysis of an avionics AFDX network. In *Proc. Of Design, Automation & Test in Europe Conference & Exhibition*, pp. 1220-1224, March 2010.

[9] J. J. Gutierrez, J. C. Palencia, and M. G. Harbour. Response time analysis in AFDX networks with sub-virtual links and prioritized switches. XV Jornadas de Tiempo Real, Santander, January-February 2012.

[10] S. Dong, Z. Xingxing, D. Lina, and H. Qiong. The design and implementation of the AFDX network simulation system. In *Proc. Of International Conference on Multimedia Technology*, pp. 1-4, October 2010.

[11] M. Tawk, X. Liu, L. Jian, G. Zhu, Y. Savaria, and F. Hu. Optimal scheduling and delay analysis for AFDX end-systems. *SAE Technical Paper*, 2011-01-2751, 2011.

[12] Y. Hua and X. Liu. Scheduling heterogeneous flows with delay-aware deduplication for avionics applications. *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 9, pp. 1790-1802, September 2012.

Task Partitioning and Priority Assignment for Hard Real-time Distributed Systems

Ricardo Garibay-Martínez, Geoffrey Nelissen, Luis Lino Ferreira, Luís Miguel Pinho
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto, Portugal
{rgmz, grrpn, llf, lmp}@isep.ipp.pt

***Abstract*—The partitioning of fixed-priority hard real-time tasks and messages in a distributed system is a well know NP-hard problem. Therefore, there are no methods that provide an optimal solution in polynomial time. In this paper, we propose the Distributed using Optimal Priority Assignment (DOPA) heuristic, which simultaneously solves the problem of assigning task to processors and assigning priorities to tasks. DOPA makes use of Audsley's Optimal Priority Assignment (OPA) algorithm to assign priorities to tasks and messages. However, in order to use the OPA algorithm for task sets with dependencies, we first transform the task set into a set of independent tasks by imposing intermediate deadlines. The experimental results show how the utilisation of the OPA algorithm increases in average the number of schedulable tasks and messages in a distributed system when compared to the utilisation of the Deadline Monotonic (DM) priority assignment usually used in other works.**

Keywords—real-time; distributed systems; task allocation; priority assignment; intermediate deadlines; holistic analysis.

I. INTRODUCTION

Real-time distributed systems are present in our everyday life. These systems range from safety critical to entertainment and domestic applications, presenting a very diverse set of requirements. Although diverse, in all these areas, modern distributed applications are becoming larger and more complex. Therefore, integrating the system's functional requirements to comply with their associated real-time constraints has shown to be a challenging problem within the real-time domain.

Hard real-time distributed systems are composed of two main elements: (i) a set of real-time *applications* and (ii) a distributed computing *platform* that executes such applications. Applications are composed of a set of tasks that communicate through messages to perform a certain functionality (e.g. realizing input/output operations, processing data, etc.). On the other hand, distributed platforms are composed of a set of processing elements (e.g. processors, ECUs, etc.) and networks, that provide the needed computational resources for tasks to be executed and messages to be transmitted.

When considering real-time applications, the processing of tasks and messages must comply with their associated time constraints. Commonly, for applications, this time constraint is expressed by an *end-to-end deadline*, which is the longest elapsed time a sequence of tasks and messages (an application) is permitted to take from the time at which it was activated until it completes its execution.

Furthermore, for a given set of applications and a given computing platform, the main objective is to find a *feasible allocation* for tasks and messages in a way that all application's end-to-end deadlines are met. Unfortunately, this problem is known to be *NP-hard* [1]. The problem of task allocation can be viewed as a two-sided problem: (i) finding the partitioning of tasks and messages onto the processing elements of the distributed system, and (ii) finding the priority assignment for tasks and messages for that partition, so that real-time tasks and messages are executed within their deadlines. Therefore, a careful trade-off between the solutions of those two subproblems needs to be taken in order to obtain a correct global solution.

Contribution. This paper presents the Distributed using Optimal Priority Assignment (DOPA) heuristic to find a feasible partitioning and priority assignment for tasks in distributed computing platforms by using the Optimal Priority Assignment (OPA) algorithm, known as Audsley's algorithm [2]. The algorithm is an optimal priority assignment algorithm for independent fixed priority tasks on uniprocessor systems. The OPA algorithm requires tasks to be independent, therefore, in order to use the OPA algorithm for task sets with dependencies (applications), we first need to transform tasks with dependencies to a set of independent tasks by imposing *intermediate deadlines*. Also, the use of intermediate deadlines makes our approach easily extensible to more powerful task models such as multithreaded parallel real-time models, when compared to previous approaches. Furthermore, our simulations show how the use of the OPA algorithm increases, in average, the number of schedulable task and messages in a distributed system, when compared to the usual method consisting in using the Deadline Monotonic (DM) priority assignment [3].

Structure of the paper. The remainder of the paper is structured as follows. Section II presents the related work, whilst Section III introduces the system model. Section IV presents the DOPA heuristic. Section V shows some experimental results, and finally, in Section VI we draw our conclusions and propose future work.

II. Related Work

The problem of task allocation is divided in two sub problems: finding the partitioning of tasks and messages onto the distributed system, and finding the priority assignment for that partition. In this section we present some relevant works that address such problems, nevertheless restraining our attention to the case of preemptive fixed-priority task scheduling based approaches.

In [4], Tindell *et al.*, addressed this issue as an optimisation problem with the general purpose *simulated annealing* algorithm. The simulated annealing algorithm is used for iterating in a random manner, over a given allocation for tasks and messages to processors and networks, and performs an evaluation based on an "energy" function, which evaluates the quality of the encountered solution (allocation). Tindell used the DM scheduling algorithm [3] to assign priorities to periodic tasks with constrained deadlines assuming that each task in an application has its own deadline and period. The latter assumption may however not always be true in real systems.

In [5], Gutierrez *et al.*, proposed an optimisation technique for the priority assignment of tasks and messages in a distributed system. They assumed a set of tasks and messages that are statically allocated to processors and networks (therefore, no partitioning phase is considered); thus, focusing on the problem of assigning the priorities to the allocated tasks and messages. Their method is based on imposing intermediate deadlines to the tasks and messages that compose a "sequence of actions" and then using DM to assign the task priorities.

The problem of partitioning tasks and messages in distributed systems is also addressed in Richard *et al.* [6]. They propose a solution based on branch-and-bound; branching (enumerating) the possible paths that can lead to and allocation, and bounding (cutting the path) whenever a feasible schedule cannot be reached by following such a path. Again, DM is used to assign the priorities assuming that each task is defined by its own deadline and period. The bounding step is performed by checking the schedulability of each branch based on the schedulability technique for RMA derived by Tindell *et al.* in [7].

In [8] and [9], the authors model the task partitioning problem as an optimisation problem. However, this work still assumes that each task has its own period and deadline, and it uses DM to assign priorities.

More recently, Azketa et al. [10] addressed the problem of task and message allocation in a distributed system by taking hand of the general purpose genetic algorithms. They use a genetic algorithm with a permutational solution encoding. They initiate their genetic algorithm by assigning priorities using the HOPA heuristic [5] which is based on DM priority assignment [3] and iterate over different solutions by applying crossover, mutation and clustering operations. To test schedulability they use the Tindell's holistic analysis [7] and Palencia's schedulability tests [11, 12].

Although there are some similarities between our method and previous works, none of the previous approaches has used Audsley's OPA to assign priorities to tasks in a distributed system. As proved in [2], the OPA algorithm is optimal for the case of preemptive fixed priority tasks, thereby implying that if the system is schedulable with DM then OPA will also find a priority assignment to schedule the task set. Further, DM has been proved to not be optimal for systems where all tasks do not release jobs simultaneously [3], which is typically the case in distributed systems due to the task precedence constraints. By adding intermediate deadlines to tasks, we show that the Audsley's algorithm can be used and that it increases the number of schedulable task sets (i.e. applications) in comparison with DM.

III. System Model

A. Real-Time Applications

A distributed real-time system is composed of software applications that we model as a set $\Gamma = \{\Gamma_1, \dots, \Gamma_n\}$ of n concurrent sequential applications Γ_i with $i \in \{1, \dots, n\}$. An application Γ_i is composed of a set $\tau_i = \{\tau_{i,1}, \dots, \tau_{i,n_i}\}$ of n_i tasks and a set of $\mu_i = \{\mu_{i,1}, \dots, \mu_{i,n_i-1}\}$ of $n_i - 1$ messages, which are executed and transmitted on processors and networks, respectively. We assume the *linear model of event-driven distributed system* [13]. In this model, each application Γ_i is activated by an external event e_i with a minimum inter-arrival time of T_i. The arrival of an external event e_i is followed by the activation of the first task $\tau_{i,1}$ of Γ_i. Whenever a task $\tau_{i,j}$ completes its execution, it sends a message $\mu_{i,j}$ to the next task $\tau_{i,j+1}$ and hence triggers its execution.

Each task $\tau_{i,j}$ is characterized by its Worst-Case Execution Time (WCET) $C_{i,j}$, and each message $\mu_{i,j}$ by a transmission time $C_{i,j}^{msg}$. Communications between tasks can be carried out within the same or different processors in the distributed system. If two tasks $\tau_{i,j}$ and $\tau_{i,j+1}$ communicate via a message $\mu_{i,j}$ and execute in the same processor, we consider that the message transmission time is negligible, thereby assuming that $C_{i,j}^{msg} = 0$. Also, an application Γ_i is characterised by an end-to-end deadline D_i, which is the longest elapsed time that the sequence of tasks and messages is permitted to take from the time in which it is activated (e_i) until it completes its execution (time at which the last task τ_{i,n_i} in the sequence of tasks and messages completes its execution). We assume that $D_i \leq T_i$. The density δ_i of an application Γ_i is given by $\delta_i = \frac{\sum_{j=1}^{n_i} C_{i,j} + C_{i,j}^{msg}}{D_i}$ and the total density of the system is defined as $\delta = \sum_{\Gamma_i \in \Gamma} \delta_i$.

We consider that tasks are scheduled with a preemptive fixed-priority algorithm. On the other hand, messages are scheduled with a non-preemptive fixed priority algorithm. We also consider that some tasks of an application can be

restrained to execute in some specific processors due to some *design constraints*. Such constraints can be related to design reasons, safety reasons, or to specific functionality offered by the processors in the distributed system (e.g. sensors, actuators, required libraries, etc.), and required for the execution of a task $\tau_{i,j}$ within an application Γ_i. Therefore, there exists a set $A \subseteq \Gamma$ of tasks that are resource constrained (they need to be executed in a specific processor), thus, those tasks are statically assigned to those processors. Also, there exist a set $\Upsilon \subseteq \Gamma$ of tasks that do not have any resource constrains and can be allocated onto any processor.

B. Distributed Computing Platform

A distributed computing platform is composed of a set of processors that provide the computing power to execute tasks, which are connected through a fixed-priority real-time network (e.g. a CAN network [14]). We assume a set $\pi = \{\pi_1, \ldots, \pi_m\}$ of m identical uniprocessor nodes for the execution of the tasks, and a single shared real-time network ϖ for message transmission.

Figure 1 shows an example of a computing platform composed of three processors and one real-time network. There are three applications Γ_1, Γ_2 and Γ_3, each composed of only one task ($\tau_{1,1}$ and $\tau_{2,1}$ and $\tau_{3,1}$). Tasks $\tau_{1,1}, \tau_{2,1}, \tau_{3,1}$ and $\tau_{5,1}$ are resource constrained (thus belonging to the set A) being pre-assigned to the specific processors 1, 2, 3 and 1 respectively. Also, there exists a list Υ of unallocated tasks $\tau_{4,1}, \tau_{4,2}$, $\tau_{5,2}$, $\tau_{6,1}$ and $\tau_{6,2}$ that can be allocated to any processor.

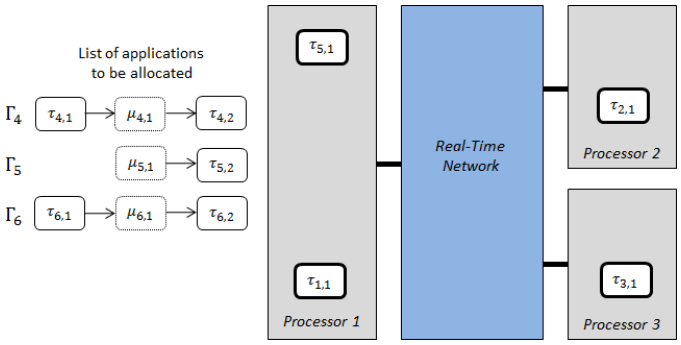


Figure 1. List of applications to be allocated in a computing platform.

The objective is to find (i) a feasible partitioning of the tasks constituting the applications onto the processors and (ii) a priority assignment to the tasks in a way that all end-to-end deadlines are met. Figure 2, shows an example of allocation α^* of tasks and messages for the unallocated applications shown in Figure 1. By looking at Figure 2 it is possible to notice that tasks in application Γ_4 ($\tau_{4,1}, \tau_{4,2}$) are allocated to the same processor, and therefore the message $\mu_{4,1}$ can be neglected, thereby reducing its communication cost to $C_{i,k}^{msg} = 0$. In the following section we present an heuristic solving this problem.

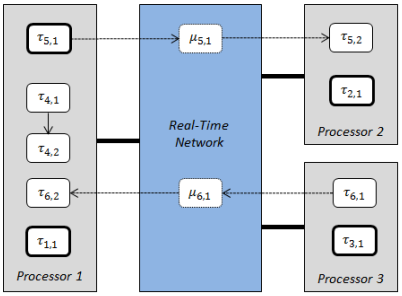


Figure 2. Example of a task allocation for the system presented in Figure 1.

IV. The DOPA Heuristic

The DOPA heuristic simultaneously addresses the two interrelated sub-problems of: (i) finding the partitioning of tasks and messages onto the elements of the distributed system, and (ii) finding the priority assignment for that partitioning.

A. Optimal Priority Assignment (OPA) algorithm

Regarding the problem of priority assignment, there exist several techniques to assign priorities to a set of preemptive independent tasks. DM [3] is the one usually considered in every work on distributed systems. DM is optimal for assigning priorities if there exists an instant in the schedule where all the tasks release a job simultaneously. However, in distributed systems tasks and/or messages have dependencies on other tasks or messages of the same application. Hence, because a task $\tau_{i,j+1}$ never starts its execution before the completion of task $\tau_{i,j}$, $\tau_{i,j}$ and $\tau_{i,j+1}$ will never release a job simultaneously, thereby violating the optimality condition of DM. One should therefore conclude that DM is not optimal for distributed systems. On the other hand, Davis and Burns [15] proved that the Audsley's OPA algorithm is optimal regarding the assignment of task priorities if there exists a schedulability test S respecting the three following conditions: (C1) the schedulability of a task $\tau_{i,j}$ according to S, may be dependent on the set of higher priority tasks $HP_{i,j}$, but not on the relative priority order of those tasks, (C2) the schedulability of a task $\tau_{i,j}$ according to a test S, may be dependent on the set of lower priority tasks, but not on the relative priority order of those tasks, and, (C3) for two tasks with adjacent priority, if their priorities are swapped, the task that has been assigned the higher priority cannot become unschedulable according to the test S, if it was schedulable at the lower priority.

```
1.  for each priority level k, lowest first{
2.     for each unassigned task τi,j{
3.        if (τi,j is schedulable at priority k according to
             test VERIFY_SCHEDULABILITY(τi,j → πk) with all
             unassigned tasks assumed to have higher
             priorities){
4.           assign τi,j to priority k
5.           break (continue outer loop)
6.        }
7.     }
8.     return unschedulable
9.  }
10. return schedulable
```

Figure 3. OPA algorithm pseudocode.

The OPA algorithm is based on three simple steps (see Figure 3): (i) check the schedulability according to S of all non-priority-assigned tasks by assuming they have the lowest priority, (ii) arbitrarily chose one that respects its deadline, and (iii) remove the chosen task from the list of non-priority-assigned tasks and start again. To verify the schedulability (VERIFY_SCHEDULABILITY($\tau_{i,j} \in \pi_k$)) of the task set, we use the schedulability analysis presented in [7]. Note however that other tests could also be used (e. g. [11, 12]). We know from [16] that the worst-case response time $r_{i,j}$ of an independent task $\tau_{i,j}$, scheduled with a preemptive fixed priority scheduling algorithm is given by the following equation:

$$r_{i,j} = C_{i,j} + \sum_{\tau_{a,b} \in HP_{i,j}} \left\lceil \frac{r_{i,j}}{T_a} \right\rceil C_{a,b} \quad (1)$$

where $HP_{i,j}$ is the set of tasks with a higher priority than $\tau_{i,j}$ that can interfere with $\tau_{i,j}$. Due to the presence of the term $r_{i,j}$ on both side of (1), this equation is usually solved in an iterative manner, $r_{i,j}^{k+1} = C_{i,j} + \sum_{\tau_{a,b} \in HP_{i,j}} \left\lceil \frac{r_{i,j}^{k}}{T_a} \right\rceil C_{a,b}$ with $r_{i,j}^{1} = C_{i,j}$.The iteration stops when $r_{i,j}^{k} = r_{i,j}^{k+1}$.

In a distributed time system, the worst-case response time $WCRT_{i,j}$ of a task $\tau_{i,j}$ can then be computed as [7]:

$$WCRT_{i,j} = r_{i,j} + \sum_{k=1}^{j-1} \left(r_{i,k} + r_{i,k}^{msg} \right) \quad (2)$$

where $r_{i,k}^{msg}$ is the response time of a message $\mu_{i,k}$ obtained with a network dependent analysis such as [14]. An application Γ_i (and hence its constituting tasks and messages) is deemed schedulable if $WCRT_{i,n_i} \leq D_i$.

Unfortunately, this schedulability test makes the schedulability of a task $\tau_{i,j}$ dependent on the response times and hence the priorities of all the other tasks and messages in Γ_i, thereby making OPA unusable. We therefore transform the tasks and messages with dependencies to an equivalent set of tasks and messages without dependencies by imposing an intermediate deadline $d_{i,j}$ ($d_{i,j}^{msg}$, resp.) to each task $\tau_{i,j}$ (each message $\mu_{i,j}$, resp.) as shown in Figure 4. The intermediate deadline $d_{i,j}$ of $\tau_{i,j}$ then becomes an offset on the release of the message $\mu_{i,j}$, and the deadline $d_{i,j}^{msg}$ of $\mu_{i,j}$, becomes an offset on the release of $\tau_{i,j+1}$. Therefore, we now have that:

$$\begin{cases} WCRT_{i,j} = d_{i,j-1}^{msg} + r_{i,j} \\ WCRT_{i,j}^{msg} = d_{i,j} + r_{i,j}^{msg} \end{cases} \quad (3)$$

implying that the worst-case response time of each task and message becomes independent of the relative priority order of higher and lower priority tasks. Now, a task $\tau_{i,j}$ (a message $\mu_{i,j}$, resp.) is deemed schedulable if $WCRT_{i,j} \leq d_{i,j}$ ($WCRT_{i,j}^{msg} \leq d_{i,j}^{msg}$, resp.) and the three Audsley's OPA algorithm validity conditions (C1, C2 and C3) are respected.

The tasks and messages intermediate deadlines are computed as a function of the application end-to-end deadline and the tasks and messages WCETs ($C_{i,j}$ and $C_{i,j}^{msg}$, respectively). For tasks and messages, the intermediate deadlines are given by:

$$d_{i,j} = d_{i,j-1}^{msg} + \frac{C_{i,j}}{\sum_{k=1}^{n_i} C_{i,k} + C_{i,k}^{msg}} D_i \quad (4)$$

$$d_{i,j}^{msg} = d_{i,j} + \frac{C_{i,j}^{msg}}{\sum_{k=1}^{n_i} C_{i,k} + C_{i,k}^{msg}} D_i \quad (5)$$

Note that from those definitions, we have that $d_{i,n_i} = D_i$. Hence, if all tasks (and messages) respect their intermediate deadlines $d_{i,j}$, i.e., $WCRT_{i,j} \leq d_{i,j}$, the end to end deadline D_i of the application is also respected.

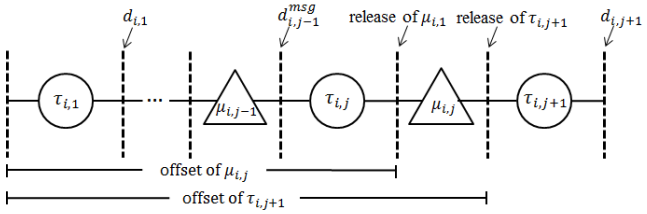


Figure 4. Intermediate deadlines.

B. Partitioning Algorithm

The problem of partitioning the task set onto the processors of the platform and assigning priorities to the tasks and messages is solved by the algorithm PARTITION(Γ) presented in Figure 5. The algorithm is entirely based on the following idea; if two successive tasks $\tau_{i,j}$ and $\tau_{i,j+1}$ of the same application Γ_i are assigned to a same processor π_k, then the message $\mu_{i,j}$ can be ommited, thereby reducing the load on the network and increasing the acceptable response time for the other tasks and messages in Γ_i. Therefore, PARTITION(Γ) tries to maximize the number of successive tasks of the same application being assigned on the same processor.

The pseudo code of the partitioning algorithm can be understood as follows. Applications $\Gamma_i \in \Gamma$ are assigned in a non-increasing density order. Tasks in Γ_i are considered in a lexical order. Each task $\tau_{i,j}$ is first assigned on the same processor than the previous task $\tau_{i,j-1}$ (if any) of the application Γ_i, thereby assuming that the message $\mu_{i,j-1}$ is unneeded and hence $C_{i,j-1}^{msg} = 0$. If the priority assignment (using OPA) does not succeed (i.e., the task is unschedulable on that processor) and the next task (if any) that must be executed in the application Γ_i has already been assigned on a processor π_k, then PARTITION(Γ) tries to assign $\tau_{i,j}$ on π_k assuming that the message $\mu_{i,j}$ is unneeded and hence $C_{i,j}^{msg} = 0$. If the priority assignment fails again, then the algorithm tries to assign $\tau_{i,j}$ on any other processor in a worst-

fit order (i.e., the processor with the smallest total density first). Finaly, the schedulability on the network is checked. Note that the intermediate deadlines of the tasks in Γ_i are recomputed at each step since their values depend on the number of messages the application must send on the network, i.e., the number of messages with $C_{i,j}^{msg} > 0$. However, this modification of the intermediate deadlines does not jeopardize the schedulability of the tasks that are already assigned to processors since, by studying Equations (4) and (5), we can see that the intermediate deadlines increase whenever a message is ommited (i.e., one of the terms $C_{i,k}^{msg}$ becomes equal to 0). Therefore, if the previous deadlines were respected, the new one will also be.

```
PARTITION(Γ)
1.   for all Γ_i ordered by non-increasing δ_i {
2.      for all τ_i,j ∈ Γ_i {
3.         assign τ_i,j to π_k| τ_i,j−1 ∈ π_k, assuming C_i,j^msg = 0
4.         call OPA_ASSIGNMENT(τ_i,j, τ_a,b ∈ π_k)
5.         if OPA succeed to assign τ_i,j {
6.            break
7.         }
8.        else if OPA fails to assign τ_i,j {
9.            assign τ_i,j to π_k| τ_i,j+1 ∈ π_k, assuming C_i,j^msg = 0
10.           call OPA_ASSIGNMENT(τ_i,j, τ_a,b ∈ π_k)
11.           if OPA succeed to assign τ_i,j {
12.              break
13.            }
14.           else if OPA fails to assign τ_i,j {
15.              for all π_k in Worst-Fit order {
16.
17.                 assign τ_i,j to π_k
18.                 call OPA_ASSIGNMENT(τ_i,j, τ_a,b ∈ π_k)
19.                 if OPA succeeds {
20.                    assign message μ_i,j to the network
21.                    VERIFY_SCHEDULABILITY(μ_i,j ∈ ϖ)
22.                    if message μ_i,j schedulable
23.                       break
24.                    else
25.                      return unschedulable
26.                 }
27.                 else
28.                    return unschedulable
29.              }
30.           }
31.        }
32.     }
32.  }
33.  return schedulable
```

Figure 5. Partitioning algorithm pseudocode.

V. EXPERIMENTAL EVALUATION

In this section we present some experimental results of our simulations of the DOPA heuristic. Let us recall that the DOPA heuristic simultaneously (i) finds the partitioning of tasks and messages onto the elements of the distributed system, and (ii) finds the priority assignment for that partitioning. For all experiments we use the `PARTITION(Γ)` algorithm for the partition of tasks and messages onto the elements of the distributed system, and we use two different priority assignment algorithms, namely DM and OPA.

One of the main objectives of this work is to demonstrate that by using the OPA algorithm, for the case of tasks with dependencies, it is possible to increase in average the number of schedulable tasks and messages in a distributed system when compared to the utilisation of the DM priority assignment, frequently used in other works.

For generating the applications Γ_i and their respective tasks $\tau_{i,j}$ and messages $\mu_{i,j}$ we follow the guidelines presented in [17] for generating random task sets for multiprocessor systems, using the Stafford's Randfixedsum algorithm [18]. The Randfixedsum algorithm generates a set of n values which are evenly distributed and whose components sum to a constant value. Thus, we use the Randfixedsum algorithm for generating unbiased sets of applications with a fixed total density $\delta_{tot} = \sum \delta_i$. For a given total density δ_{tot}, the algorithm returns n applications with density δ_i; with values from a minimum density bound $\delta_i^{min} = 0.1$ for each application, and a maximum density bound $\delta_i^{max} = 0.9$. For generating the tasks' and messages' densities we use again the Randfixedsum algorithm taking as an input the previous generated densities $\delta_i = \sum(\delta_{ij} + \delta_{i,j}^{msg})$, obtaining a set of values $\delta_{i,j} = d_{i,j}/T_i$ for tasks and $\delta_{i,j}^{msg} = d_{i,j}^{msg}/T_i$ for messages with values from a minimum density bound for tasks and messages $u_{i,j}^{min} = 0.01$ and a maximum density of $u_{i,j}^{max} = 0.9$. The WCETs of tasks $C_{i,j}$, messages $C_{i,j}^{msg}$ and end-to-end deadlines D_i are generated as recommended in [17]; we considered that applications have implicit end-to-end deadlines ($D_i = T_i$) following a uniform distribution. For each experiment 100 sets are generated.

Figure 6 (a) shows the number of accepted tasks sets over 100 experiments for different total densities δ_{tot}. We simulate 50 applications that execute tasks and transmit messages in a computing platform of 10 processors and 1 network. It is possible to see that OPA in average performs better in terms of the number of accepted task sets. For example, the OPA algorithm accepts 52% of task sets with total system density of 9. In contrast, the DM algorithm reaches 16% with the same system density.

In Figure 6 (b) we show the number of accepted task sets for 100 experiments simulating 50 applications that execute tasks and transmit messages in a computing platform composed of 1 network and a varying number of processors. The density is fixed to $\delta_{tot} = 8$. It is possible to see that OPA in average performs better, for example, when the number of processors is equal to 9, the OPA algorithm accepts 70% of task sets, whilst the DM algorithm only accepts 30% of task sets.

Figure 6 (c) shows the number of accepted tasks sets over 100 experiments, where we vary the number of applications with a fixed total density $U_{tot} = 8$ to be scheduled in a computing platform of 10 processors and 1 network. It is possible to see that the OPA algorithm, in average performs better; in the range between 10 and 50 applications, OPA always accepts more task sets than DM. For example, for the case of 40 applications, the OPA algorithm accepts 69% of task sets, in contrast the number of accepted tasks sets obtained by the DM algorithm is 34%. Note that the number of accepted task sets increases with the number of generated applications. This behaviour can be explained by the fact that the average density of tasks and messages decreases, thereby meaning that more tasks can be scheduled on each processor in average.

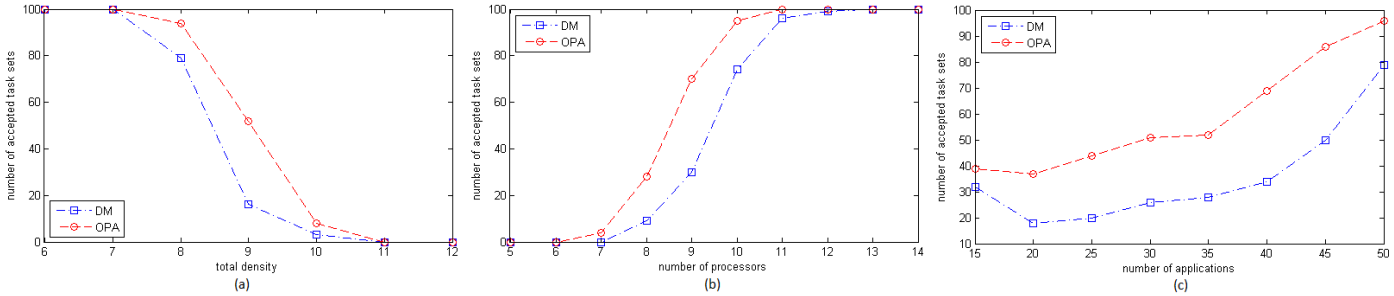


Figure 6. 100 experiments varying (a) the total density, (b) the number of processors, and (c) the number of applications in the system.

The effects presented in Figures 6 (a), (b), (c), can be explained because, when DM is used for assigning priorities, it fails more often than OPA due to its non-optimality. Therefore, such non-schedulable tasks need to be partition onto other processor in the distributed system, thus increasing the number of messages in the network, which leads to an increasing number of unschedulable systems.

VI. Conclusions and Future Work

This paper presented the DOPA heuristic for the simultaneous partitioning and priority assignment of tasks and messages (applications) onto the constituting elements of the distributed system by using the OPA algorithm known as Audsley's algorithm [2].

We proposed a method that imposes intermediate deadlines to tasks and messages thus permitting the use OPA for tasks with dependencies. Furthermore, our approach is easily extensible to more powerful task models such as multithreaded parallel real-time models, when compared with other works addressing sequential dependent tasks and messages in a distributed system. We demonstrated through simulations that OPA increases, in average, the number of schedulable tasks and messages in a distributed system, when compared to the DM algorithm, when using the same partition algorithm.

We are currently working on the extension of the DOPA heuristic for considering multithreaded parallel real-time models and the inclusion of more complex topologies of communication networks.

Acknowledgments

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects FCOMP-01-0124-FEDER-015006 (VIPCORE) and FCOMP-01-0124-FEDER-037281 (CISTER); by FCT and EU ARTEMIS JU, within project ARTEMIS/0003/2012, JU grant nr. 333053 (CONCERTO); by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/71562/2010.

References

[1] A. Burns, "Scheduling hard real-time systems: a review," *Software Engineering Journal,* vol. 6, no. 3, pp. 116-128, 1991.

[2] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," University of York, Department of Computer Science, 1991.

[3] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation,* vol. 2, no. 4, pp. 237-250, 1982.

[4] K. Tindell, A. Burns and A. J. Wellings, "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems,* vol. 4, no. 2, pp. 145-165, 1992.

[5] J. J. Gutiérrez-García and M. González-Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proceedings of the Third IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1995.

[6] M. Richard, P. Richard and F. Cottet, "Allocating and scheduling tasks in multiple fieldbus real-time systems," in *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03)*, 2003.

[7] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming,* vol. 40, no. 2-3, pp. 117 - 134 , 1994.

[8] W. Zheng, Q. Zhu, M. Di Natale and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proc. 28th IEEE International Real-Time Systems Symposium (RTSS'2007)*, 2007.

[9] A. Metzner and C. Herde., "Rtsat--an optimal and efficient approach to the task allocation problem in distributed architectures," in *Proc. of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006.

[10] E. Azketa, J. P. Uribe , J. J. Gutiérrez, M. Marcos and L. Almeida, "Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems," in *In Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM*, 2011.

[11] J. C. Palencia and M. González-Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of The 19th IEEE Real-Time Systems Symposium (RTSS'98)*, 1998.

[12] J. C. Palencia and M. Gonzalez-Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, 1999.

[13] J. C. Palencia, J. J. Gutiérrez and M. González-Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proc. of IEEE Ninth Euromicro Workshop on Real-Time Systems*, 1997.

[14] R. I. Davis, S. Kollmann, V. Pollex and F. Slomka, "Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways," *Real-Time Systems,* vol. 49, no. 1, pp. 73-116, 2013.

[15] R. I. Davis and A. Burns , "Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems," in *Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, 2009.

[16] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal,* vol. 29, no. 5, pp. 390-395, 1986.

[17] P. Emberson, R. Stafford and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *In 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'10)*, 2010.

[18] R. Stafford , "Random vectors with fixed sum," [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/9700, 2006.

Run-time Support for Real-Time Multimedia in the Cloud

Tommaso Cucinotta[(1)], Karsten Oberle[(2)], Manuel Stein[(2)] Peter Domschitz[(2)], Sape Mullender[(3)]
[(1)] Bell Laboratories, Alcatel-Lucent, Dublin, Ireland
[(2)] Bell Laboratories, Alcatel-Lucent, Stuttgart, Germany
[(3)] Bell Laboratories, Alcatel-Lucent, Antwerp, Belgium
E-mail: firstname.lastname@alcatel-lucent.com

Abstract

This paper summarizes key research findings in the area of real-time performance and predictability of multimedia applications in cloud infrastructures, namely: outcomes of the IRMOS European Project, addressing predictability of standard virtualized infrastructures; Osprey, an Operating System with a novel design suitable for a multitude of heterogeneous workloads including real-time software; MediaCloud, a novel run-time architecture for offering on-demand multimedia processing facilities with unprecedented dynamism and flexibility in resource management.

The paper highlights key research challenges addressed by these projects and shortly presents additional questions lying ahead in this area.

1 Introduction

The continuous evolution of computation and communication technologies is causing a paradigm shift in our own idea of computing. Indeed, the widespread availability of broadband connections is simply leading to the end of the Personal Computer era, marking the beginning of a new era where computing is mostly distributed. Users not only recur to "the network" to retrieve contents. They also store and manage their data remotely, keeping it accessible from a variety of heterogeneous devices and widespread locations. Users exhibit increasingly challenging requirements on the computing capabilities remotely accessible, not limiting themselves to delegate off-line computations to remote servers, but rather expecting more and more *interactive and real-time* applications to be readily available on-demand. This is witnessed by the increasing use of on-line collaborative document editing or video authoring services, for example.

Being a major driver to the Cloud Computing model, a key role in the new panorama is being played by *virtualization*. With the possibility to host multiple virtualized machines seamlessly onto the same physical hardware, the possibility to create virtual network overlays abstracting away from the actual network topology, and the possibility to dynamically live-migrate virtualized machines while they are running, virtualization technologies constitute an enabler for flexible and efficient management of physical resources in data centers.

However, an application domain where the provisioning of interactive on-line services with nearly "real-time" responsiveness remains challenging from a technical viewpoint is the domain of *multimedia*. Indeed, multimedia contents are characterized by an *isochronous* delivery model, where for example audio or video frames need to be delivered at perfectly regular intervals. However, the network over which most of these contents are distributed nowadays, the Internet, has not been designed with predictability in mind. Furthermore, often multimedia servers that need to deliver contents to many users concurrently make use of software technologies (e.g., Operating System, middleware, etc.) that have been designed for best-effort performance, not for predictable execution. Even more, the use of multimedia compression algorithms leads to a naturally fluctuating networking and computing workload that is usually reflected in variable execution and transmission times. Last, but not least, the use of virtualization technologies increases further the unpredictable behaviors in the execution of services, as due to the increased degree of sharing of physical resources (particularly computing and networking) among different (often heterogeneous) applications. The overall outcome

is an *irregular*, *randomly varying* and *unpredictable* delivery of multimedia contents to end users, making it very difficult to adhere to precise QoS specifications in Service Level Agreements (SLAs) [13].

2 Related Work

The problem of guaranteeing stable Quality of Service levels to cloud and distributed applications has been investigated on multiple levels.

The performance implications of data movements have received a lot of attention in the cloud environment, e.g., for proximity reasons [27] and bulk data migration purposes [16]. Placement of computations in large distributed clouds was hypothetically evaluated in [9]. When dealing with deployments spanning geographically distributed data centers, it has been proposed [24] to consider network requirements for the selection of computing locations across the WAN under various scenarios. In [30], authors show the benefits of considering the network topology and overall demand for response times when load-balancing workloads across neighboring data centers. In [6], it is proposed to leverage end-to-end application-level latency expression specifications for optimal placement across geographically distributed locations. In [3], a placement algorithm is proposed that finds a mapping for components of an application with a minimal diameter of the spanned network graph.

Concerning the isolation of virtualized software on the computing level, authors proposed [20] to use an EDF-based scheduling algorithm [21] for Linux on the host to schedule Virtual Machines (VMs). Unfortunately, the proposed scheduler is built into a user-space process (VSched), leading to unacceptable context switch overheads. Furthermore, VSched cannot properly guarantee temporal isolation in presence of a VM that blocks and unblocks, e.g., as due to I/O. IRMOS has improved over these approaches (see Section 3).

Some authors investigated [14] the performance isolation of virtual machines, focusing on the exploitation of various scheduling policies available in the Xen hypervisor [8]. Furthermore, various enhancements to the Xen credit scheduler have been proposed [12] to address various issues related to the temporal isolation and fairness among the CPU share dedicated to each VM. Adaptive CPU allocation has been proposed [23] to maintain a stable performance of VMs, using application-specific metrics to run the necessary QoS control loops.

Concluding, while various solutions have been proposed to the problem of performance isolation in virtualized environments, these are either not focused on critical parameters that are necessary for running real-time applications, or they lack of a proper low-level real-time scheduling infrastructure, which is needed for supporting temporal isolation among concurrently running software components. The following section explains how IRMOS addressed these issues.

3 IRMOS/ISONI Platform

The IRMOS European Project[1] has investigated on how to enhance execution of real-time multimedia applications in distributed virtualized infrastructures. The IRMOS Intelligent Service-Oriented Networking Infrastructure (ISONI) [28, 24] acts as a Cloud Computing IaaS provider, managing and virtualizing a set of physical computing, networking and storage resources available within a provider domain. One of the key innovations introduced by ISONI is its capability to ensure guaranteed levels of resource allocation for individual hosted applications. In ISONI, each distributed application is specified by a Virtual Service Network (VSN), a graph whose vertexes represent Application Service Components (ASCs), deployed as VMs, and whose edges represent communications among them. VSN elements are associated with precise computing and networking requirements. These are fulfilled thanks to the allocation and admission control logic pursued by ISONI for VM instantiation, and to the low-level mechanisms shortly described in what follows. A comprehensive ISONI overview is out of the scope of this paper and can be found in [28, 24].

Isolation of Computing. In order to provide scheduling guarantees to individual VMs scheduled on the same system, processor and core, IRMOS incorporates a deadline-based scheduler [7] for Linux. It provides temporal isolation among multiple possibly complex software components, such as entire VMs. It uses a variation of the CBS algorithm [1], based on EDF, for ensuring that each group of processes/threads is scheduled on the available CPUs

[1] Interactive Real-time Multimedia Applications on Service-oriented Infrastructures. More information is available at: `http://www.irmosproject.eu`.

for a specified time every VM-specific period.

Isolation of Networking. Isolation of the traffic of independent VMs within ISONI is achieved by a VSN-individual virtual address space and by policing the network traffic of each deployed VSN. The virtual addresses overlay avoids unwanted crosstalk between services sharing physical network links. Mapping individual virtual links onto diverging network paths allows for a higher utilization of the network infrastructure by mixing only compatible traffic classes under similar predictability constraints and by allowing selection of more than just the shortest path. Traffic policing avoids that the network traffic going through the same network elements causes any overload leading to an uncontrolled growth of loss rate, delay and jitter for the network connections of other VSNs. It is important to highlight that ISONI allows for the specification of the networking requirements in terms of common and technology-neutral traffic characterization parameters, such as the needed guaranteed average and peak bandwidth, latency and jitter. An ISONI transport network adaptation layer abstracts from technology-specific QoS mechanisms of the networks, like Differentiated Services [5], Integrated Services [32, 31] and MPLS [25]. The specified VSN networking requirements are met by choosing the most appropriate transport network, among the available ones. More detailed information on QoS provisioning between data centers within an ISONI domain is given in [29]. Other interesting results from the research carried out in IRMOS include algorithms for the optimum placement of distributed virtualized applications with probabilistic end-to-end latency requirements [18], a probabilistic model for dealing with workload variations in elastic cloud services [17] and the use of neural networks for estimating the performance of VM execution under different scheduling configurations [19]. The effectiveness of IRMOS/ISONI has been demonstrated, among others, through an e-Learning demonstrator [10].

4 Ongoing and Future Work

The IRMOS project has addressed various challenges in the area of predictable execution of virtualized multimedia applications. However, a number of problems still remain unaddressed. For example, these workloads would benefit from lighter run-time environments than VM instances containing full-fledged OSes, as used in current cloud infrastructures. These are among the motivations of MediaCloud [11] and Osprey [26], two projects from Bell Labs described below.

MediaCloud. Handling the predicted growth of video and media traffic is one of the key challenges future generation networks need to address. Up to now, cache-assisted delivery schemes [15] enabled the networks to scale with the data traffic imposed by video centric services. However, video delivery is becoming more tailored to the specific user accessing it (e.g., user-specific ads). Moreover, future video centric media services will see more people actively producing content. Also, the area of on-line gaming has a growing interest in providing highly dynamic and interactive multimedia. With more contents dynamically produced, customized and accessed from mobile devices, intermediate processing of media streams will need an unprecedented degree of dynamism and adaptability that go beyond the possibilities of today's virtualized infrastructures.

Indeed, the contemporary cloud computing model is based on virtual machines that are statically allocated ahead of time, before it is known who accesses which contents and from where. Furthermore, only relatively small and infrequent adjustments can be done dynamically, as due to the unavoidable "inertia" behind migration of VMs, whose contained OSes often amount to GB of data for the OS volatile memory and tens of GB for the VM disk image. In consequence, today's applications are typically designed in a way, that data has to be moved through the network to where the application is executed [27] which proves costly for live multimedia contents. We believe that this paradigm will change in the future, meaning that an intelligent infrastructure will also force the movement of applications in the line of data and demand sources. Therefore we are working on ways to optimize the delivery of (real-time) media services on top of a distributed cloud environment.

The MediaCloud Project [4] is investigating novel virtualized computing paradigms specifically tied to multimedia applications, where the location of media processing can be quickly altered at runtime, when sources and destinations of the multimedia applications are known. Moving towards a largely distributed service execution paradigm requires software to be split up into fine-grained ser-

vice components. Designing a service from a plurality of atomic service components requires an online set-up of how those components interact, that is, which media flows the components exchange at service run-time. The customer of such a service should not need to care about the location of execution in the network. MediaCloud takes care of finding best-fit resources during service run-time, when sources and sinks of relevant media streams are known, resulting in reduced end-to-end service latencies and offloaded networks by keeping traffic local. The execution framework ensures fluent media flow forwarding between service components. This deferred allocation puts the foundation for very efficient management of resources. However, one of the main challenges to address is the instantiation of the required media processing functions that needs to be performed so quickly as to not impact the QoE for the end users. The achievement of such a goal is severely obstructed by the use of machine virtualization. Investigations and experiments have shown that using fully-fledged operating systems inside a virtual machine as execution containers can hardly offer the required performance, scalability and efficiency for running distributed real-time media-centric services [4].

MediaCloud introduces a lightweight execution container design, which is fully optimized for supporting efficient execution of fine-grained service components. These can be added and deleted and media flows can be moved between, added to or removed from components at run-time. Such dynamic mechanisms in combination with the ability to move service components between execution resources in the network during run-time, build the basic foundation for an efficient, top-performing and scalable service execution on distributed processing resources in the network.

MediaCloud introduces a novel flow driven execution environment optimized for the processing of media functions, which departs from traditional software stacks being deployed in today's virtualized cloud infrastructures.

Preliminary measurements [11] performed on the prototype implementation proved that MediaCloud is able to provide the envisaged level of agile resource allocation and utilization. It supports instantiation of media processing functions, as well as re-assignment of media processing components across processing resources, in the time-frame of 2 to 3 milliseconds, in some investigated scenarios.

Even highly optimized VM-based systems can accomplish these tasks in seconds but not in milliseconds. Additional investigations indicate that MediaCloud is also able to achieve much more efficient resource utilization. A collection of cooperative media processing tasks executed on a MediaCloud controlled processing resource consumed only about half of the resources needed when doing the same job by making each task a process on the Linux OS. At the same time, we could show significantly better end-to-end service delay figures for a collection of media processing components executed on MediaCloud despite its lower resource utilization.

Osprey. As discussed above, while bringing a number of advantages in terms of ease of (and seamless) management of software, machine virtualization in itself is also constituting the root cause of many technically unnecessary overheads in today's cloud applications. Indeed, virtualized infrastructures have replicated software layers providing similar functions, such as resources management and allocation (e.g., CPU scheduling, memory and peripheral management). Also, many attempts to reduce such overheads so as to obtain a smarter resource management among the hypervisor and the hosted guest OSes usually result in the increase of the degree of para-virtualization of the guest OSes, reducing the advantages of full machine virtualization (e.g., seamless server consolidation and increased isolation/security).

As a consequence, we claim that more attention should be devoted to *OS virtualization* instead, a technique allowing for a single Operating System to create multiple isolated "domains", where independent software can be deployed. For example, the Linux LXC project[2] and FreeBSD Jails[3] provide such a mechanism. However, even though applying QoS-aware (or real-time) resource management techniques in a General-Purpose OS (GPOS) is principally possible, as shown in IRMOS by patching the Linux kernel with a real-time scheduler [7], nonetheless this leads to a suboptimal solution from a number of viewpoints. Still, we keep having replicated functionality among the hypervisor and guest OSes. Furthermore, there are resource wastes due

[2]More information is available at: `http://lxc.sf.net`.

[3]More information is available at: `http://www.freebsd.org/doc/en\_US.ISO8859-1/books/handbook/jails.html`.

to the unawareness of the host and guest schedulers, i.e., in order to guarantee certain real-time performance levels, more resources need to be allocated than strictly needed, because of the hierarchical composition of schedulers [2]. Furthermore, a GPOS is designed for a relatively low number of processes/cores and tasks to handle. However, a big server in a virtualized data center may easily include tens/hundreds of cores in a single machine. A nowadays GPOS does not have the necessary degree of scalability and flexibility in configuration that allow for an efficient management of resources in these conditions.

Osprey [26] is a new OS under development at Bell Laboratories suitable for a multitude of future computing scenarios, including: embedded systems; cloud-hosted real-time multimedia applications with tight timing requirements and highly fluctuating and horizontally scalable resource requirements; future data-intensive and high-performance applications. Osprey includes mechanisms for scalable, low-overhead and energy-aware resources management and scheduling, supporting predictable execution. The OS can be deployed with a very small memory footprint and a lightweight set of functionality, so as to fit within embedded devices dealing with multimedia (e.g., smart phones, set-top boxes, smart TVs, etc...), and very fast boot-up times, so to reduce energy-consumption due to stand-by modes. Osprey can be deployed within network elements, such as base stations, routers, firewalls. In cloud computing environments, Osprey is suitable both for thin clients and for provider-side run-time environments for future cloud applications. It includes OS-level virtualization, and an OS architecture featuring a very small micro-kernel, just capable of switching between address spaces and fielding system calls, traps and interrupts. It uses asynchronous communication primitives among core OS components and for user-kernel space interactions, reducing unneeded overheads. Also, it includes into the core OS mechanisms for check-pointing, migration and recovery of processes, enabling fault-tolerance.

Finally, Osprey integrates Pepys [22], a novel networking protocol for content distribution, with native and efficient support for named replicated contents and mobile users. It also avoids unneeded copies of data across the network stack, enabling high-performance data-intensive applications.

5 Conclusions

In this paper, key research efforts in the area of real-time performance and predictability for multimedia applications in cloud infrastructres have been summarized, along with some of the research challenges that deserve further attention, and a short overview of ongoing research projects promising to address these challenges.

References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[2] L. Abeni and T. Cucinotta. Efficient virtualisation of real-time activities. In *Proceedings of the IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2011)*, Irvine, CA, December 2011.

[3] M. Alicherry and T.V. Lakshman. Network aware resource allocation in distributed clouds. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications*, Orlando, Florida, USA, March 2012.

[4] M. Bauer, S. Braun, and P. Domschitz. Media processing in the future internet. In *Proc. of EuroView 2011: Visions of Future Generation Networks*, Wuerzburg, Germany, July 2011.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *RFC2475, An Architecture for Differentiated Service*. IETF, Dec 1998.

[6] F. Chang, R. Viswanathan, and T. L. Wood. Placement in clouds for application-level latency requirements. In *Proceedings of the 5th IEEE International Conference Cloud Computing*, Honolulu, Hawaii, USA, June 2012.

[7] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari. Hierarchical multiprocessor CPU reservations for the linux kernel. In *Proceedings of the 5^{th} International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009)*, Dublin, Ireland, June 2009.

[8] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of 3 CPU schedulers in Xen. *SIGMETRICS Perform. Eval. Rev.*, 35:42–51, September 2007.

[9] K. Church, A. Greenberg, and J. Hamilton. On delivering embarrassingly distributed cloud services. In *Proceedings of the Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII)*, Calgary, CA, October 2008.

[10] T. Cucinotta, F. Checconi, G. Kousiouris, K. Konstanteli, S. Gogouvitis, D. Kyriazis, T. Varvarigou, A. Mazzetti, Z. Zlatev, J. Papay, M. Boniface, S. Berger, D. Lamp, T. Voith, and M. Stein. Virtualised e-learning on the irmos real-time cloud. *Service Oriented Computing and Applications*, pages 1–16, 2011. 10.1007/s11761-011-0089-4.

[11] P. Domschitz and M. Bauer. Mediacloud - a framework for real-time media processing in the network. In *Proceedings of EuroView 2012*, Wuerzburg, Germany, July 2012.

[12] G. Dunlap. *Scheduler development update*. Xen Summit Asia, Shanghai, 2009.

[13] G. Gallizo, R. Kuebert, G. Katsaros, K. Oberle, K. Satzke, S. Gogouvitis, and E. Oliveros. A service level agreement management framework for real-time applications in cloud computing environments. In *Proceedings of the 2nd International ICST Conference on Cloud Computing (CloudComp 2010)*, Barcelona, Spain, October 2010.

[14] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX International Conference on Middleware*, pages 342–362, New York,USA, 2006. Springer-Verlag New York, Inc.

[15] M. Hofmann and L. Beaumont. *The book about content networking*. Morgan Kaufman, Feb 2005. ISBN I 55860 834 6.

[16] D. Klein, M. Menth, R. Pries, Phuoc Tran-Gia, M. Scharf, and M. Sollner. A subscription model for time-scheduled data transfers. In *Integrated Network Management (IM), 2011 IFIP/IEEE Internat. Symp. on*, pages 555–562, May 2011.

[17] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou. Admission control for elastic cloud services. In *Proc. of the 5th IEEE International Conference on Cloud Computing*, Honolulu, Hawaii, USA, June 2012.

[18] K. Konstanteli, T. Cucinotta, and T. Varvarigou. Optimum allocation of distributed service workflows with probabilistic real-time guarantees. *Service Oriented Computing and Applications.*, 4:68:229–68:243, December 2010.

[19] G. Kousiouris, T. Cucinotta, and T. Varvarigou. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *Journal of Systems and Software*, In Press, Corrected Proof:–, 2011.

[20] B. Lin and P. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of the IEEE/ACM Conference on Supercomputing*, November 2005.

[21] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 20:46–61, January 1973.

[22] S. Mullender, P. Wolkotte, F. Ballesteros, E. Soriano, and G. Guardiola. Pepys – the network is a file system. Technical Report TR RoSaC20114, Bell Labs and Rey Juan Carlos University, 2011.

[23] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds. In *Proceedings of the 5th European Conference on Computer systems (EuroSys)*, Paris, France, April 2010.

[24] K. Oberle, M. Kessler, M. Stein, T. Voith, D. Lamp, and S. Berger. Network virtualization: The missing piece. In *Intelligence in Next Generation Networks, 2009. ICIN 2009. 13th International Conference on*, pages 1–6, October 2009.

[25] E. Rosen, A. Viswanathan, and R. Callon. *RFC3031, Multi-protocol Label Switching Architecture*. IETF, Jan 2001.

[26] J. Sacha, J. Napper, H. Schild, S. Mullender, and J. McKie. Osprey: Operating system for predictable clouds. In *in Proceedings of the 2nd Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV'12)*, Boston, MA, USA, June 2012.

[27] B. Tiwana, M. Balakrishnan, M. Aguilera, H. Ballani, and Z. M. Mao. Location, location, location! modeling data proximity in the cloud. In *In HotNets IX: Ninth Workshop on Hot Topics in Networking*, pages 1–6, Monterey, CA, October 2010.

[28] T. Voith, M. Kessler, K. Oberle, D. Lamp, A. Cuevas, P. Mandic, and A. Reifert. *ISONI Whitepaper v2.0*, 2009.

[29] T. Voith, K. Oberle, and M. Stein. Quality of service provisioning for distributed data center inter-connectivity enabled by network virtualization. *Elsevier Future Generation Computer Systems (FGCS 2011)*, 2011.

[30] I. Widjaja, S. Borst, and I. Saniee. Geographically distributed datacenters with load reallocation. In *DIMACS Workshop on Cloud Computing*, Piscataway, NJ, USA, December 2011.

[31] J. Wroclawski. *RFC 2210, The Use of RSVP with IETF Integrated Services*. IETF, Sep 1997.

[32] J. Wroclawski. *RFC2211, Specification of the Controlled Load Quality of Service*. IETF, Sep 1997.

Resource Management for Service Level Aware Cloud Applications

Cristian Klein,
Francisco Hernández-Rodriguez
Department of Computer Science
Umeå University, Sweden

Martina Maggio,
Karl-Erik Årzén
Department of Automatic Control
Lund University, Sweden

Abstract

Resource allocation in clouds is mostly done assuming hard requirements, time-sensitive applications either receive the requested resources or fail. Given the dynamic nature of workloads, guaranteeing on-demand allocations requires large spare capacity. Hence, one cannot have a system that is both reliable and efficient.

To mitigate this issue, we introduce service-level awareness in clouds, assuming applications contain some optional code that can be dynamically deactivated as needed. We propose a resource manager that allocates resources to multiple service-level-aware applications in a fair manner. To show the practical applicability, we implemented service-level-aware versions of RUBiS and RUBBoS, two popular cloud benchmarks, together with our resource manager. Experiments show that service-level awareness helps in withstanding flash-crowds or failures, opening up more flexibility in cloud resource management.

1. Introduction

Cloud computing radically changed the management of data-centers [5]. In the past, machines used to have one specific purpose. The need for a new functionality, such as a new web application, implied the purchase of a new Physical Machine (PM). This tendency resulted in poor resource utilization and energy waste. This issue was further aggravated by the growing number of cores per PM, driven by the end of frequency scaling, which increased the amount of unused hardware per node. However, thanks to advances in cloud computing technologies, applications are now wrapped inside Virtual Machines (VMs) and consolidated onto fewer PMs [20].

As a result, resource management becomes a key issue. Specifically, it is crucial to decide how the available capacity is distributed among applications to ensure that on-demand resource requests are satisfied given the available hardware. In this area, there has been a tremendous amount of work, mostly assuming that applications are time-sensitive – lengthy responses may lead to dissatisfied users – and their resource requirements are not flexible – the application is either given the needed amount of resources or fails. Combined with the fact that most cloud applications have dynamic resource requirements [23], this imposes a fundamental limitation to cloud computing, which decrease its flexibility: To guarantee on-demand resource allocations, the data-center needs large spare capacity, leading to inefficient resource utilization.

For increased resource management flexibility, we propose introducing *Service-Level (SL) awareness* in clouds. SL aware applications are characterized by a dynamic parameter, the *service-level*, that monotonically affects both the end-user experience, as well as the computing capacity required by the application. For example, online shops offer end-users recommendations of similar products they might be interested in. No doubt, recommender engines greatly increase user experience. However, due to their sophistication, they are highly demanding on computing resources [18]. By selectively activating or deactivating the corresponding code, proportionally to the service-level, resource consumption can be controlled and data-center overload can be avoided at the expense of end-user experience.

SL awareness opens up the possibility to deal predictably and efficiently with unexpected events. Unexpected peaks — also called flash crowds — may increase the volume of requests by up to 5

This work was partially supported by the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control and through the LCCC Linnaeus Center. Also, we received partial support from the EU project VISION Cloud under grant agreement no. 257019, the Swedish Government's strategic effort eSSENCE and from the ELLIIT Excellence Center.

times [3]. Similarly, unexpected failures reduce the capacity of the data-center until they are repaired. Also, unexpected performance degradations may arise due to interference among co-located applications [20]. These phenomena are well-known and software is readily written to cope with them, using techniques such as replication and dynamic load balancing, as long as resource provisioning is sufficient [2, 15]. However, given the short duration of such unexpected events, it is often economically unfeasible to provision enough capacity for them. On the contrary, using SL awareness, the infrastructure can simply ask applications to temporarily reduce their requirements. Consequently, end-user experience is reduced, since the optional code is not executed. However, delivering partial content in a timely manner is better than overloading the data-center and rendering hosted applications unresponsive.

In this article we build the necessary software infrastructure to support SL-aware cloud applications. We focus on the resources of a single PM, leaving multiple-PM extensions for future work. We assume that the application developer followed the guidelines to produce SL-aware application presented in Section 2. We propose a Resource Manager (RM) that coordinates the resource allocation among applications competing for the same resources (Section 3). The highlight of our contribution is that the design is backed up by theoretical results from game theory. Our system provides specific guarantees on desirable properties such as convergence and fairness among the applications, which translates to withstanding capacity shortages predictably. We evaluate our approach, in Section 4, using two well-known cloud benchmark applications, RUBiS [24] and RUBBoS [4], that are extended with SL-aware recommender engines. To foster further research and pursue repeatability we have made all source code publicly available[1].

2. Application model

In this section we describe the application model that we expect developers to follow. We assume that every application i is composed of time-sensitive requests, which have to be executed before a soft deadline expires: Exceeding it should be minimized, to avoid user dissatisfaction. As an example, such applications can be made Service-Level (SL)-aware, by marking a part of the request as *optional*. Being able to run optional computations is desirable, as they would improve end-user experience, however, deactivating them is preferred to missing a deadline. Let the probability of executing optional computations between time[2] k and $k+1$ be equal to the SL of the application s_i^k. Consequently, the capacity required by the application is proportional to s_i^k.

Every application is requested to regularly update the Resource Manager (RM) about its performance. More precisely, a **matching value** respecting three properties should be computed. First, the matching value should be close to zero when the assigned resources are perfectly matched with the current SL of the application. Second, if the matching value is positive, the resources assigned to the application are abundant and the application can compute at a higher SL, or the amount of assigned resources can be reduced. Third, and dual, if the matching value is negative, either more resources have to be provided or the application should reduce its SL to avoid missing deadlines.

For the application model described above, we chose to compute our matching value f_i^k as follows:

$$f_i^k = 1 - t_i^k / \bar{t}_i \tag{1}$$

where $\bar{t}_i$ is the desired deadline and t_i^k is the maximum response-time of requests served from $k-1$ to k. The matching value f_i^k is the only value that the application has to communicate to the RM. It is easy to prove that our choice respects the properties described above.

Our framework can exploit the adaptivity of applications that change their SL to offer an overall better performance. Each adaptive application i may change the SL it runs at, as a function g_i of the current performance, called the **update rule**. At time j the application i updates its service level according to

$$s_i^{j+1} = g_i(s_i^j, f_i^j) \tag{2}$$

that can be different for each application. This internal feedback loop belongs to the application and the RM is not informed about its behavior, nor about its execution interval (the distance between j and $j+1$). Examples of how to design such loop can be found in [17]. As a result, both the SL s_i^j and the update rule g_i are private to the application, i.e., the RM is not informed about them. This assumption allows the RM to run in linear time with respect to the number of applications, resulting in a lower overhead compared to a complex optimization approach where the RM also selects the SLs of the applications. Moreover, this allows applications to customize their definition of the SLs and their update rule. Two proposals for update rules are described in [17]. Note that our framework allows application to be non-cooperative, i.e, SL-unaware, as

[1] GitHub repository: https://github.com/cristiklein/cloudish

[2] Throughout, time is assumed discrete and denoted with k or j, while i always represents the application.

most existing applications are. If no matching value is communicated, the RM simply assumes it to be zero.

To clarify the above concepts, we sketch an e-commerce website as an example of an SL-aware application. We consider the visualization of a product's page as one request. The optional code of such a request consists in retrieving recommendations of similar products. For each request, besides retrieving the product information, the application runs the recommender engine with a probability s_i^j. Increasing s_i^j increases the amount of served recommendations, thus increasing end-user experience, but also the capacity requirements of the application. To avoid saturation, the application is made self-adaptive by controlling the parameter s_i^j so as to keep the maximum response-time around a configured deadline.

One of the main differences between this work and similar research in the context of embedded systems [7, 19] is that we do not assume anything about the application's behavior, thus, the RM does not have access to the SL update rules. In fact, our framework is completely general with respect to the choice of g_i.

3. Resource management

The role of the RM is to select the capacity of the Physical Machine (PM) that each application is allowed to use. In many works cited in Section 5, cloud resource allocation is done based on *monitored* resource usage. However, this approach cannot be used to support SL-aware applications. For example, when an application's CPU usage is low, without additional information, the RM cannot distinguish whether the application is abundantly provisioned and runs at maximum SL, or insufficiently provisioned but runs at low SL to compensate. Therefore, our RM does not directly monitor the resource usage of the applications but uses information on the applications' performance that are conveyed through the matching value defined in Eq. (1)[3] without needing to know the SLs of the applications.

Let us now describe the RM's behavior. We denote with $c_i^k \in [0,1]$ the capacity assigned at time k to the i-th application relative to the total capacity C of the PM. At initialization, the RM sets the capacities to $c_i^0 = 1/n$ where n is the number of applications. Subsequently, at the beginning of each control interval, it first retrieves measurements for all the matching values f_i^k — as defined in Eq. (1) — then updates each capacity according to

$$c_i^{k+1} = c_i^k - \varepsilon_{rm}\left(f_i^k - c_i^k \cdot \sum_p f_p^k\right) \tag{3}$$

where ε_{rm} is a design constant. Given that initially $\sum_i c_i^0 = 1$, one can prove through induction that:

$$\sum_i c_i^k = 1 \tag{4}$$

i.e., the RM enforces that the total allocated capacity does not exceed the available one. Since the matching values of the applications are closer to zero when the resources they receive match their SLs, the new allocation favors the applications that are more distant from their target performance values — whose matching values are more negative. The new resource allocation reflects the relative distance between the applications' performance. Finally, the computed relative capacities c_i^k are multiplied by the total capacity C, to obtain the absolute values C_i^k. The RM itself needs to make sure that it gets enough resources to function correctly, either by reserving some capacity for itself, or by running with a higher priority than the applications. The RM's complexity is **linear** with respect to the number of applications, which allows its implementation to have low overhead.

Let us summarize the convergence analysis of the designed system; detailed proof can be found in [7, Section IV]. Using game-theory and treating applications as players bidding for resources, it can be shown that the RM allocations converge to a **stationary point**, that is characterized by the following property: Applications are either performing sufficiently well, which means that their matching values are close to zero, or are poorly performing but already operate at minimum service level. It was also proven that if a stationary point where all the matching values of the running applications are driven to zero exists, this point is reached. Moreover, the RM ensures **fairness** among applications. Whenever the applications have similar definition for their matching values, the framework theoretically guarantees that, in case of overload, the resources assigned to the applications converge to equal values. In other words, applications contribute equally to dealing with the overload.

4. Experimental evaluation

Experimental setup. Our testbed is a single PM equipped with two AMD Opteron™ 6272 processors[4] and 56 GB of memory, which hosts several Virtual Machines (VMs). We used Xen 4.1.2 as a hypervisor and Ubuntu 12.04.2 LTS 64-bits with Linux kernel

[3] As long as the matching value respects the three introduced properties, its formulation can be changed.

[4] 2100 MHz, 16 cores per processor, no hyper-threading

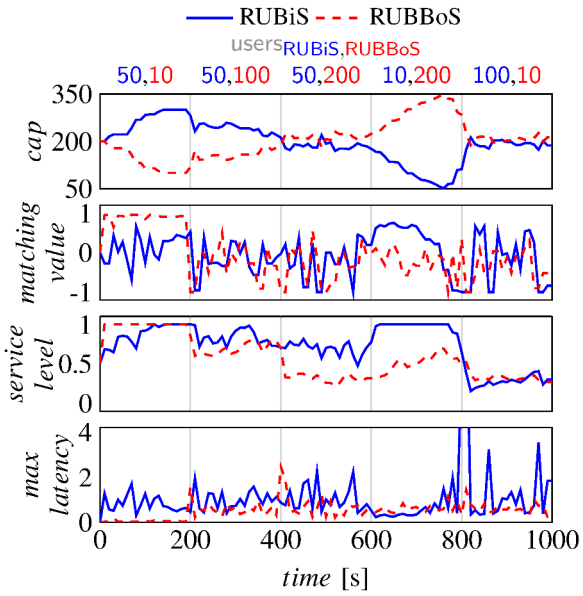


Figure 1: Resource manager and two applications.

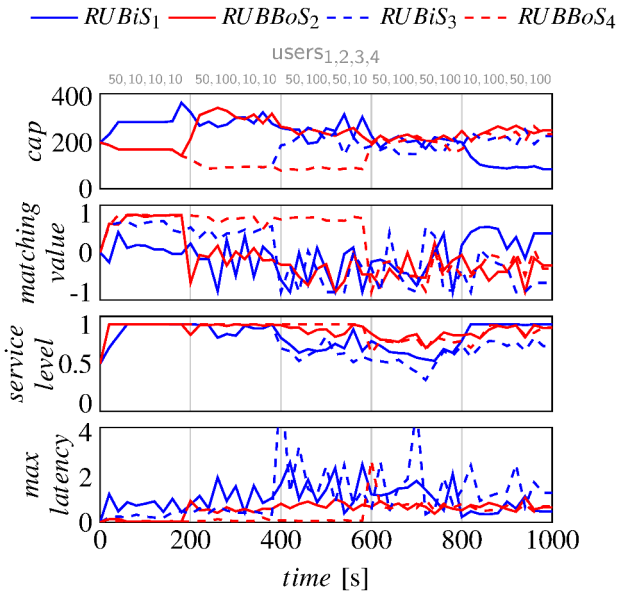


Figure 2: Resource manager and four applications.

version 3.2.0, both for the privileged Dom_0 and the unprivileged Dom_U VMs. Every unprivileged VM is configured with 4 GB of memory and a variable number of virtual CPUs. The number of virtual CPUs is determined as a function of the **cap** parameter — a cap of 400 means that the VM has exclusive access to 4 cores of the PM, while with $cap = 50$ the VM has access to a single core of the PM, but only for half of the time. We deployed our SL-aware versions of RUBiS and RUBBoS, each inside a single VM among Dom_U, and the RM inside Dom_0. Each application's VM contains the self-adaptive version of the application and all tiers belonging to it — Apache web server, PHP interpreter, MySQL server. Since we focus on CPU allocations, we ensured that the database could be fully cached in memory.

Experimental methodology. To simulate the users' behavior, we dynamically select a think-time and a number of users. Each user runs an infinite loop, which waits for a random time and then issues a request. The random waiting time is chosen from an exponential distribution, whose rate is given by the think-time parameter. Since we are interested in studying how well the framework controls CPU resources, we made sure that network or disk did not influence our results. Therefore, we ran our workload generator inside Dom_0 on a dedicated core. Furthermore, we disabled logging and made sure that each VM had enough memory to keep the whole database in-memory. Indeed, disk activity measured during the experiments was negligible. The RUBiS and RUBBoS applications are made SL-aware as described in [17], with desired deadlines of 1 and 0.5 seconds, respectively.

The platform is limited to 4 cores of the PM, on which we deploy both the SL-aware RUBiS and RUBBoS. Their caps are selected by the RM, as described in Eq. (3), based on the matching values they send, computed according to Eq. (1). The RM's control period is set to 5 seconds and ε_{rm} is 0.2. During the experiments, we vary the number of users accessing the two services at time 200, 400, 600 and 800, and observe the behavior of the RM and applications.

Results. Figure 1, displaying the results, is structured as follows. Four metrics are plotted as a function of time for each of the two applications: the cap chosen by the RM, the matching value, the SL and the maximum user-perceived latency. The vertical bars represent time intervals during which the number of users is kept constant, with values listed on top. At time instant 0, the experiment starts in its default configuration: Each application is allocated half of the platform and both SLs are 0.5. Since the load on RUBBoS is low, it increases the SL to maximum. Similarly, the adaptive RUBiS will try to increase the SL, however, it has insufficient resources to do so immediately. The RM detects this conditions, through the transmitted matching value, and rebalances the platform, so as to reduce RUBBoS's cap and increase RUBiS's cap. Thanks to this, the system reaches a configuration in which both applications may run at maximum SL. At time instant 200, we increase the number of RUBBoS users. RUBBoS reacts to avoid overload and reduces

the SL. Furthermore, the RM increases its cap and decreases RUBiS's cap. RUBiS reduces its SL to deal with the new resource allocation. Thus, the system approaches a stationary point, in which the performance requirements of both applications are satisfied. Indeed, both RUBiS and RUBBoS users experience maximum latencies around the configured desired deadline of each application (1 second and 0.5 seconds). Similarly, new stationary points are reached after the changes in number of users occurring at 400, 600 and 800.

To further test the fairness of the system, we conducted an experiment with 4 SL-aware applications, 2 RUBiS and 2 RUBBoS VMs, and a platform consisting of 8 cores. As can be seen in all intervals of Fig. 2, applications that do not run at full SL are assigned equal caps, whose value we call *fair cap*. In other words, despite targeting different desired deadlines and executing different code, applications that reduce their SL to deal with the infrastructure's overload contribute with an equal amount of resources to overload reduction. This is easily observed for application 1, 2, 3 and 4 in the 4th interval, whose caps settle around 200 or applications 2, 3, 4 in the 5th interval, whose caps settle around 230. Some applications may be able to run at full SL with fewer resources than the fair cap. For these applications, their cap is reduced to the minimum value which allows them to run at full SL. Thus, such applications contribute with even more resources to overload reduction, without sacrificing their SL. For example, application 1 in the 5th interval runs at full SL with a cap around 98, which is smaller than the fair cap of 230.

Note that in both Figs. 1 and 2, latencies may temporarily increase above the desired deadline. This is expected, since applications continuously try to maximize their service-level, hence, latencies may shortly overshoot. To conclude, we experimentally showed that the RM behaves as theoretically designed, avoiding overload while respecting fairness among applications.

5. Related work

Managing resources in clouds is a challenging task. Resource management schemes are either application or infrastructure-centric. Performing *application-centric* resource allocation (e.g. [6, 8, 26]) means deciding the right amount of resources to allocate avoiding under- or over-provisioning. However, applications are not cooperative and cannot reduce their requirements if resources are congested. In this way, the limitations of the underlying infrastructure are neglected, taking only the application's point-of-view.

Application-centric allocation can be combined with game theory. For example, Ardagna et al. [1] studies resource allocation in which users bid for resources and the provider sets the price to maximize his revenue. A solution which converges to a Nash equilibrium is proposed. Sharma et al. [25] proposes Kingfisher, a system that tries to minimize the cloud tenant's deployment cost while reacting to workload changes. However, none of these works take into account the capacity limitations of the cloud provider.

Although some works deal with performance differentiation for multiple classes of clients [21], to our knowledge, the only cloud application that comes close to being SL aware is Harmony [9]. It adjusts the consistency-level of a distributed database as a function of the incoming end-user requests, so as to minimize resource consumption. This is a specific example of SL awareness in cloud applications, and the adaptation strategy is not reflected in the resource allocation.

Infrastructure-centric resource allocation strategies like [12, 27] mostly regard applications as non-cooperative "black-boxes", with hard resource requirements. Among the different contributions to the area, we most closely relate to those dealing with over-subscription (also called over-booking) [28]. In [11, 22] the RM is assumed to know the minimum application requirements a priori, which is not a valid assumption in a cloud environment. In [16], application requirements are modeled as random variables and statistical analysis is applied to avoid data-center overload. In [14] the approach is extended with correlation coefficients between the requirements and portfolio theory is used to increase over-subscription, while controlling the overload risk. However, in both of these works no remedy is given to overload conditions, besides having to pay a penalty to the user. A possible solution is presented in [29] by allowing the provider to suspend the least "important" VMs. However, this solution may be unacceptable when the VMs are hosting interactive, Internet-facing applications.

SL-awareness can be an alternative or a complement to other techniques. For example, *out-scaling* is often proposed as a solution to temporary lack of capacity [13] — requesting VMs from a public cloud provider, such as Amazon EC2 or Rackspace, effectively creating a *hybrid cloud*. SL awareness can be an initial, temporary solution, during the time interval when out-scaling is set up, or an alternative, whenever out-scaling is not an option such as budget constraints or privacy concerns. In fact, with out-scaling, besides the cost for renting the VMs, the owner would also have to pay the cost of transferring her data onto the public cloud and back into the data-center after the unexpected condition expired. Also, the owner

may deal with sensitive data, such as company know-how, credit card transactions, user profiles, that are not transferable outside the private data-center. Finally, cloud providers themselves have limited capacity and even Amazon EC2 — one of the largest computing inventories — can run out of capacity [10].

To the best of our knowledge, this is the first work that deals with SL-aware cloud applications, integrating them with resource allocation. Existing papers either do not study how such applications change their SL and interact with the infrastructure or how the infrastructure coordinates multiple such applications.

6. Conclusion

In this paper we discussed a proposal for resource allocation to service-level aware cloud applications. We proposed a game-theoretic resource manager to coordinate the demands of multiple applications in a predictable and fair way. These applications can reduce the burden they inflict on the cloud infrastructure, therefore cooperating to the better management of the available resources, in particular to avoid data-center overload. We implemented the framework and tested it with real-life experiments, demonstrating that we allocate resources fairly to the running applications.

References

[1] D. Ardagna, B. Panicucci, and M. Passacantando. "A game theoretic formulation of the service provisioning problem in cloud systems". In: *WWW*. 2011.

[2] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[3] P. Bodik et al. "Characterizing, modeling, and generating workload spikes for stateful services". In: *SOCC*. 2010.

[4] *Bulletin Board Benchmark*. URL: http://jmob.ow2.org/rubbos.html.

[5] R. Buyya et al. "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility". In: *Future Generation Computer Systems* 25.6 (2009).

[6] E. Caron, F. Desprez, and A. Muresan. "Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients". In: *J. Grid Comput.* 9.1 (2011), pp. 49–64.

[7] G. Chasparis et al. "Distributed Management of CPU Resources for Time-Sensitive Applications". In: *ACC*. 2013.

[8] L. Y. Chen et al. "Achieving application-centric performance targets via consolidation on multicores: myth or reality?" In: *HPDC*. 2012.

[9] H.-E. Chihoub et al. "Harmony: Towards Automated Self-Adaptive Consistency in Cloud Storage". In: *CLUSTER*. 2012.

[10] Compute Cycles. *Lessons learned building a 4096-core Cloud HPC Supercomputer*. Mar. 2011. URL: http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html.

[11] T. Cucinotta et al. "On the Integration of Application Level and Resource Level QoS Control for Real-Time Applications". In: *Industrial Informatics, IEEE Transactions on* 6.4 (2010), pp. 479–491.

[12] A. Fedorova, M. Seltzer, and M. D. Smith. "Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler". In: *PACT*. 2007.

[13] A. J. Ferrer et al. "OPTIMIS: A holistic approach to cloud service provisioning". In: *Future Generation Computer Systems* 28.1 (2012), pp. 66–77.

[14] R. Ghosh and V. K. Naik. "Biting Off Safely More Than You Can Chew: Predictive Analytics for Resource Over-Commit in IaaS Cloud". In: *CLOUD*. 2012.

[15] J. Hamilton. "On designing and deploying internet-scale services". In: *LISA*. 2007, pp. 1–12.

[16] I. Hwang and M. Pedram. "Portfolio Theory-Based Resource Assignment in a Cloud Computing System". In: *CLOUD*. 2012.

[17] C. Klein et al. *Introducing Service-level Awareness in the Cloud*. Tech. rep. ISRN LUTFD2/TFRT-7641-SE. Lund University, July 2013.

[18] J. A. Konstan and J. Riedl. "Recommended to you". In: *IEEE Spectum* (Oct. 2012).

[19] M. Maggio et al. "A Game-Theoretic Resource Manager for RT Applications". In: *ECRTS*. 2013.

[20] J. Mars et al. "Bubble-Up: increasing utilization in modern warehouse scale computers via sensible co-locations". In: *MICRO*. 2011.

[21] A. Merchant et al. "Maestro: quality-of-service in large disk arrays". In: *ICAC*. 2011.

[22] R. Rajkumar et al. "A resource allocation model for QoS management". In: *RTSS*. 1997.

[23] C. Reiss et al. "Heterogeneity and Dynamicity of Clouds at Scale". In: *SOCC*. 2012.

[24] *Rice University Bidding System*. URL: http://rubis.ow2.org.

[25] U. Sharma et al. "A Cost-Aware Elasticity Provisioning System for the Cloud". In: *ICDCS*. 2011.

[26] U. Sharma, P. Shenoy, and D. F. Towsley. "Provisioning multi-tier cloud applications using statistical bounds on sojourn time". In: *ICAC*. 2012.

[27] Z. Shen et al. "CloudScale: elastic resource scaling for multi-tenant cloud systems". In: *SOCC*. 2011.

[28] L. Tomas and J. Tordsson. "Improving Cloud Infrastructure Utilization through Overbooking". In: *CAC*. 2013.

[29] L. Wang, R. Hosn, and C. Tang. "Remediating Overload in Over-Subscribed Computing Environments". In: *CLOUD*. 2012.