Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

EVALUATION OF COMPUTING RESOURCE MANAGEMENT METHODS FOR SDR CLOUDS

Vuk Marojevic, Ismael Gomez, Antoni Gelonch
Radio Communications Research Group, Universitat Politècnica de Catalunya
{marojevic|ismael.gomez|antoni}@tsc.upc.edu

ABSTRACT

SDR clouds centralize the digital signal processing resources of base stations and employ cloud computing technology enabling computing resource sharing and on demand resource allocation. Each new user session request requires the allocation of resources from the computing resource pool for executing the corresponding transmitter and receiver waveforms. An efficient computing resource management is essential for complying with the tight timing constraints of wireless communications services. Different solutions exist. This paper reviews two resource management concepts—*global-dynamic* and *partitioned-static scheduling*. We introduce complexity models for analyzing the viability of these two approaches in the context of SDR clouds. The results show that global-dynamic scheduling may incur significant resource overheads.

1. INTRODUCTION

SDR clouds describe distributed antenna systems that connect to a data center employing cloud computing technology. The data center will perform the digital signal processing tasks of base station transceivers. A single SDR cloud data center may cover a metropolitan area with millions of inhabitants. Thousands of processors will then serve thousands of user sessions in parallel [11].

SDR clouds are scalable and provide additional degrees of flexibility for improving the resource efficiency. The SDR cloud is long-term vision. We envisage its gradual development and deployment, which can be divided into three phases:

1. Centralized baseband processing,
2. Automatic computing resource allocation and management,
3. Virtualization, enabling computing resource sharing.

The consolidation of these phases will enable the realization of SDR clouds. The centralized baseband processing concept is being tested already, supported by major operators and infrastructure providers. Automatic computing resource allocation and management is necessary for efficiently managing complex computing systems in dynamic environments. Virtualization will finally allow the independent management of SDR cloud customers, sharing a common resource pool in a fair and controlled way.

Wireless communications transceivers process user signals in several processing stages. The digital signal processing chain defines the SDR transmitter or receiver functionality and is called a waveform. Many waveforms synchronously process continuous data streams for as long as the wireless communications session lasts. Samples are regularly generated by the ADC at the receiver and are fed to the DAC at the transmitter. The ADC/DAC works at a specific sampling frequency, which determines the necessary processing speed for real-time service provisioning. Waveforms are thus data driven and often modeled as directed acyclic graphs (DAGs) with hard real-time processing and data flow requirements. Modern waveforms are very processing intensive and, generally, need to be executed in a distributed fashion [1].

The SDR cloud data center represents a large-scale distributed computing system. Each new user session request implies allocating resources from a shared computing resource pool for executing the transmitter and receiver waveform of the new user while other users' waveforms may already be running. A session termination correspondingly frees resources. Hence, distributed computing resources will be dynamically occupied and released. The tight timing constraints of wireless communication services require developing efficient computing resource management approaches that operate in real time while introducing as little resource overhead as possible. Different approaches exist. This paper reviews two general resource management concepts: *global-dynamic* and *partitioned-static scheduling*.

Global scheduling is the most commonly employed scheme for running general purpose applications on multicore processors. The literature on this topic is vast, covering many different scheduling types. Some algorithms support (hard or soft) real-time applications. A dynamic global scheduler executes periodically and may allow process preemption. All processes that are ready for execution are organized in a common queue. The scheduler periodically decides which process is executed when and where. The

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

scheduling overhead is then given by the scheduling periodicity and the execution time per scheduler invocation. It can be significant [2].

Although the global scheduler is not designed for signal processing applications, this family of schedulers is employed in many modern modems, from multi-core mobile phones to base stations. IBM's wireless network cloud (WNC) uses a global scheduler. The processing workload is therefore divided into several threads, each processing a set of OFDM symbol blocks. The global scheduler dynamically distributes threads among cores. This way waveforms are allocated to multi-core processors running Linux with real-time kernel extensions [3].

The alternative to global scheduling is partitioned scheduling, where a mapping algorithm first distributes the waveform modules among the processing elements (PEs) followed by local scheduling. *Partitioned-static scheduling* in the context of SDR clouds then runs the mapper and scheduler once for each new user session request without rescheduling executing waveforms. The overhead is then a function of the user arrival rate and the resource allocation complexity per user.

This paper analyzes the complexity of these two computing resources management methods for assessing their suitability for SDR clouds. Section 2 provides a brief overview of some main scheduling concepts. Section 3 introduces the complexity models and metrics for analyzing the overheads of the two scheduling methods in medium and highly loaded scenarios. Section 4 derives the conclusions.

2. SCHEDULING

Scheduling refers to the process of temporal organization of events. In the computing context, the scheduler essentially determines the execution intervals of processes. Although processing time is often considered as the most critical resource, the access to other types of computing resources (communication buses, memories, etc.) needs to be scheduled as well. Scheduling is a complex field of research in with several decades of innovation. This section can only provide a sketchy overview of the different scheduling concepts.

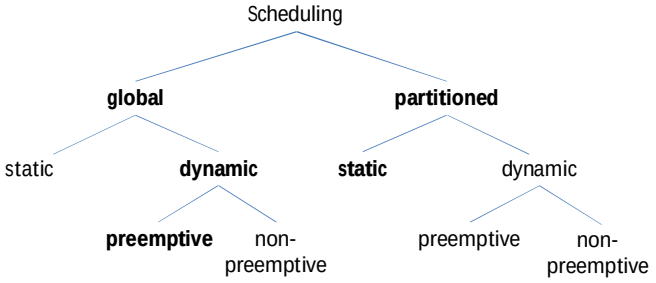

Fig. 1. Scheduling classification.

First of all we may distinguish between uniprocessor and multiprocessor scheduling. The uniprocessor scheduling problem reduces to determining the execution order of processing tasks on a single processing element (PE). Distributed computing systems require multiprocessor scheduling, which will play an increasingly important role in the future. The trend goes towards many-core devices, which can provide better computing and energy figures [6].

Multiprocessor scheduling is, except for the trivial cases, an NP complete optimization problem [8]. In simple terms this means that searching for an optimal solution is computationally intractable. Many suboptimal approximations and heuristics were therefore introduced.

Scheduling algorithms fall under different categories and several taxonomies, such as [5], were proposed. Figure 1 indicates some major scheduling classes, which we discuss in continuation.

2.1. Global versus Partitioned Scheduling

A global scheduler allocates computing resources to computing tasks using a single global queue. A single scheduler thus determines the execution start of processes, data flows, and so forth for all tasks in the system. Global scheduling is the most popular scheduling approach, deployed by the real-time (RT) Linux kernel, among others. Popular algorithms are the global earliest deadline first (G-EDF) and the global rate monotonic scheduler (G-RMS) [7]. Global scheduling directly addresses the multiprocessor scheduling problem and can achieve better schedules, especially if application speedup it the objective. Since the multiprocessor scheduling problem is NP complete, any solution will be suboptimal, though.

The partitioned scheduler works in two phases: It globally maps tasks to PEs, and locally schedules the process execution on each PE. Partitioned schedulers cannot provide optimal schedules, in general. Moreover, if tasks are added to the system at runtime, it may be necessary to repartition the entire system.

Hybrid solutions are currently investigated. These are often referred to as clustered or semi-partitioned scheduling, where tasks are first assigned to groups of processors (clusters) and then scheduled by global schedulers, one per cluster. This allows for combining the benefits of global and partitioned scheduling.

2.2. Static versus Dynamic Scheduling

Real-time scheduling algorithms can be static or dynamic. Static algorithms assign processes to processors and determine the execution start of each process a priori, before the execution start (offline). They are often used to schedule periodic tasks. Aperiodic tasks whose characteristics are not know a priori need to employ dynamic scheduling algorithms.

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

Dynamic scheduling assumes very little a priori knowledge about the application's resource needs and the execution environment. Whereas in the static case schedules can be computed before the application is ever executed, dynamic scheduling decisions are not made until a process begins its life in the dynamic environment [5].

Dynamic scheduling causes runtime overhead, which can be significant for certain applications. Applications composed of fine grain tasks will lead to more active tasks and, thus, a higher scheduling overhead. Shorter task execution times incur more overhead, which grows with the number of PEs in the system [10].

2.3. Preemptive versus Nonpreemptive Scheduling

Dynamic schedulers can be preemptive or nonpreemptive. Preemptive schedulers can stop a process in execution for executing another process instead. The stopped process will be resumed at some later time instant and possibly migrated to another processor. This provides full flexibility for adapting to frequent changes in the workload and managing tasks with different priorities.

Nonpreemptive schedulers cannot stop an executing process for scheduling another. Later arriving tasks then need to be assigned to the remaining resources or *scheduling holes* or must wait until more resources become available. Nonpreemptive schedulers may accept fewer tasks in a dynamic system as deadlines (of later arriving tasks) may be missed. They incur less system overhead, because of no task migrations and fewer context switches. Nonpreemptive schedulers are appropriate for scheduling tasks of equal priorities and applications with regular execution characteristics.

3. ANALYSIS

This section derives simple models for comparing the complexity of a partitioned static scheduler (PSS) with a global dynamic preemptive scheduler (GDPS). We apply our analysis on a processing cluster of an SDR cloud data center. A processing cluster is a group of PEs that processes user signals associated to group of radio cells. Clustering is a practical technique for limiting the resource management complexity while still enabling resource sharing [11].

3.1 Schedulers

A) PSS

The synchronous and regular data flow on average of many radio access technologies (RATs), including UMTS and LTE, enables assuming a pipelined execution pattern. Pipelined execution, where all waveform tasks execute each time slot, eliminates the precedence constraints between tasks and greatly simplifies the scheduling and synchronization processes. The time slot should be specified as a fraction of a millisecond for ensuring low processing latencies [11].

The PSS implicitly models the computing resource *time*. A feasible mapping thus ensures meeting the real-time processing requirements with the available computing resources. The scheduler determines the execution order so that the data processing and propagation finished within the time slot boundaries [1]. The computing resource matrices are dynamically updated so that each new session request can access only the remaining computing resources (processing power and interprocessor bandwidths). This eliminates resource conflicts and enables static scheduling in the SDR cloud context.

B) GDPS

The GDPS is commonly deployed in distributed computing contexts because of its flexibility to adapt to varying workloads and computing conditions. Without loss of generality, we assume that the GDPS needs to recalculate the schedule of all active waveforms each time a user enters or exits the system. This assumption can be easily relaxed for contemplating different scheduler invocation rates.

3.2. Complexity Models

The scheduling overhead, that is, the resources used for running the scheduler, is proportional to the complexity of each scheduling invocation and the invocation rate. The PSS maps only the new waveform to the available computing resources during the user session initiation phase. The processor-internal scheduling has a complexity of $O(1)$ and can be neglected. The GDPS, on the other hand, reschedules all active waveforms at each scheduling event.

The parameters of our models are summarized in Table I. We assume that the PSS and the GDPS have the same complexity t per waveform. That is,

$$t_1 = A \cdot n \text{ [s]} \tag{1}$$

and

$$t_2 = B \cdot m \cdot n^2 \text{ [s]}, \tag{2}$$

depending if the waveform is executed as an indivisible processing block ($t = t_1$) or as a processing chain of m processing blocks ($t = t_2$). Independent tasks (one per waveform) are scheduled in the first case. This essentially involves choosing one out of n PEs for executing any waveform. A waveform modeled as a processing chain of m processes allows for distributed processing and higher resource occupation, in general. The real-time data flow between processes, however, needs to be scheduled to the available interprocessor communication resources and explains the complexity increase of (2) when compared to (1).

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

Table I – Modeling parameters.

t	Execution time of the PSS or GDPS per waveform [s/waveform]
A, B	Scaling factors [s]
n	Number of PEs in the cluster
m	Number of waveform tasks
λ	User arrival rate, that is, average number of new user session requests per second [users/s]
u	Number of active user sessions (including new session requests)

Equation (2) corresponds to the complexity order of the t_1-mapping algorithm applied to tasks graphs with precedence constraints [1]. When the precedence constraints are dropped, the complexity order reduces to $O(m \cdot n)$, from which (1) follows. Most practical schedulers have complexity orders that can be modeled as (1), (2), or somewhere in between: $t_1 \leq t \leq t_2$ [4] [7] [9].

The scaling factors A and B translate from complexity orders to execution times. Their values can be obtained from measuring the scheduler execution time on the PE that will run the scheduler. We assume only one waveform per user here, although two independent waveforms, one for the transmitter and one for the receiver, would need to be scheduled for each user. The scaling factors may absorb this.

The time between consecutive session establishments follows a Poisson distribution with a mean of λ^{-1}. In other words, λ^{-1} represents the average time between session requests and λ the average user arrival rate.

We model the number of active users per cluster as

$$u = \rho \cdot n / k, \tag{3}$$

where k indicates the fraction or fractional multiple of a PE needed for executing a waveform in real time. A value of k = 1.5, for instance, means that the processing power equivalent to 1.5 PEs is needed for executing a waveform. This assumes homogeneous waveforms and PEs. Parameter ρ specifies the average system load. A value of ρ = 0.5, for instance, indicates a 50 % occupation of the cluster processing resources.

A) PSS

The overhead of the PSS can be directly obtained as the complexity of allocating resource for a single waveform times the mean user arrival rate:

$$PE\text{-}overhead_{\mathrm{PSS}} = \lambda \cdot t. \tag{4}$$

The $PE\text{-}overhead_{PSS}$ represents the fraction or fractional multiple of a single PE needed for executing the PSS. Assuming that the scheduler runs on the same computing cluster as the waveforms, the resource it occupies are not available for digital signal processing. We may divide this overhead by the number of PE in the cluster to obtain the fraction of the cluster processing power dedicated to scheduling:

$$CL\text{-}overhead_{\mathrm{PSS}} = \lambda \cdot t / n. \tag{5}$$

The dependency on the different modeling parameters can be easily found by combining (5) with (1) or (2): The $CL\text{-}overhead_{\mathrm{PSS}}$ is

- proportional to λ,
- independent of m ($t = t_1$) and proportional to m ($t = t_2$), respectively, and
- independent of n ($t = t_1$) and proportional to n ($t = t_2$), respectively.

Being independent of n is highly desirable because this means that the scheduling complexity is independent of the cluster size. In this case we may not even need to define clusters if other parameters—λ, in particular—would not increase with the cluster size and increasing geographical coverage. A scheduler whose complexity is proportional to n is usually said to scale well with the number of PEs. We will analyze this further in Section 3.3.

B) GDPS

The GDPS recalculates the schedule of all active tasks at each scheduling event, which here coincides with a user session initiation or termination. We assume that sessions are initiated and terminated independently and analyze the complexity in stable operation, where the average number of users entering and leaving the system is balanced. The percentage of a PE dedicated to scheduling can then be modeled as

$$PE\text{-}overhead_{\mathrm{GDPS}} = 2\lambda \cdot u \cdot t. \tag{6}$$

The processing overhead on cluster basis is then

$$CL\text{-}overhead_{\mathrm{GDPS}} = 2\lambda \cdot u \cdot t / n. \tag{7}$$

Combining (7) with (1) or (2) and (3) we conclude that the $CL\text{-}overhead_{\mathrm{GDPS}}$ is

- proportional to λ,
- proportional to ρ,
- independent of m ($t = t_1$) and proportional to m ($t = t_2$), respectively, and
- proportional to n ($t = t_1$) and n^2 ($t = t_2$), respectively.

Assuming modular waveforms, the GDPS does not scale that well with the number of PEs.

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

Table II – Simulation parameters.

A, B	0.25 µs [1]
m^*	10, 20
n	4, 8, 12, 16, 24, 32, 40, 48, 56, 64
λ	0.1, 1, 10
ρ	0.5, 0.8
k	0.7

* Applies to $t = t_2$ (2).

3.3. Results

Table II indicates the parameter values chosen for the simulations. We simulate different traffic demands, cluster sizes and waveform granularities. Figure 2 shows the cluster processing overhead as a function of the number of PEs n for different user arrival rates λ and a single processing block per waveform ($t = t_1$).

As expected, we observe that the cluster processing overhead of the PSS is independent of n. The complexity of the GDPS increases with the number of PEs and is considerably higher than that of the PSS. However, the overhead of the GDPS is approximately 0.02 % for 64 PEs and λ = 10, occupying only 1.3 % of the processing resources of a single PE. This overhead is perfectly assumable, meaning that the GDPS is apt for working under the given conditions.

The results of Fig. 2 assume a coarse-grained scheduler, which schedules waveforms as indivisible processing blocks. This will generally impede achieving high processing loads. The loss in processing power can be as high as 50 % for waveforms requiring just over 50 % of a PE's capacity. High processing loads can be achieved when assuming smaller processing blocks and distributing the execution of each waveform [11]. The next three figures consider this case.

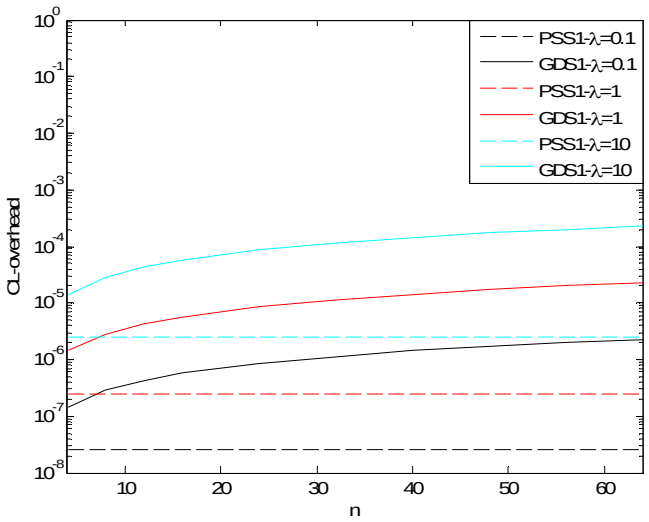

Fig 2. Cluster processing overhead due to (1) for ρ = 0.5.

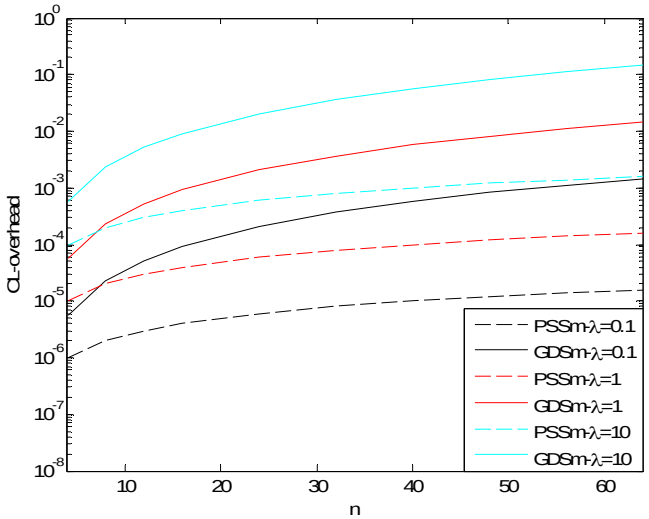

Fig 3. Cluster processing overhead due to (2) for m = 10, ρ = 0.5.

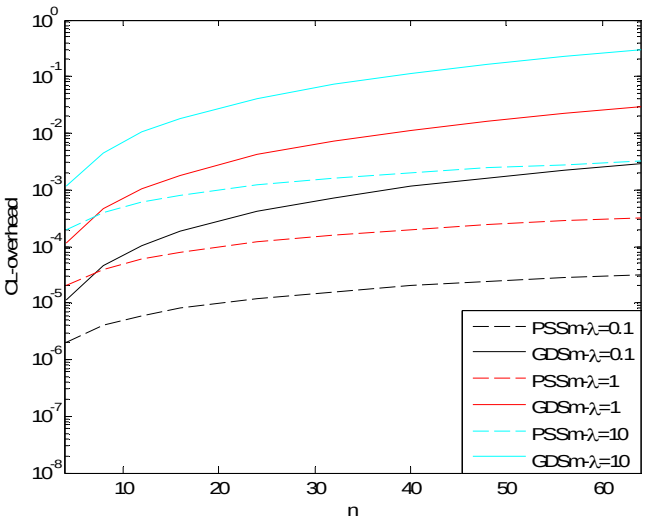

Fig 4. Cluster processing overhead due to (2) for m = 20, ρ = 0.5.

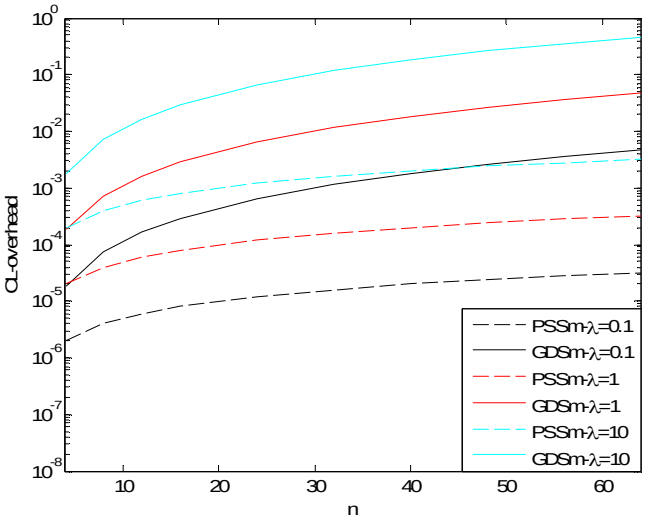

Fig 5. Cluster processing overhead due to (2) for m = 20, ρ = 0.8.

Proceedings of SDR-WInnComm 2013, Copyright © 2013 Wireless Innovation Forum All Rights Reserved

Figure 3 shows that scheduling precedence-constrained tasks results in a significant complexity increase. The complexities of the PSS and the GDPS both increase with the number of PEs. The dependency on n is much more significant for the GDPS, as indicated already in Section 3.2.B. A cluster processing overhead of approximately 15 %, which is obtained for the GDPS with n = 64 and λ = 10, may be inacceptable. It means that 0.15 · 64 = 9.6 PEs are dedicated to scheduling, leaving only 54.4 PEs for digital signal processing.

When doubling the number of waveform tasks from 10 (Fig. 3) to 20 (Fig. 4), the scheduling overhead also doubles. Figure 5 finally confirms that the $CL\text{-}overhead_{GDPS}$ is a function of the average system load. This is so because the more users are active, the more tasks will need to be scheduled at each scheduling invocation. This is different for the PSS, which determines the resource allocation only for the waveform associated with the new user session request. The dashed curves are thus identical in Figs. 4 and 5.

Our results confirm those of [10] mentioned earlier: The scheduling overhead increases

- with the task granularity (m),
- with the number of PEs (n), and
- inversely to the task execution time (λ^{-1}).

If we limit the scheduling overhead to 1.5 %, which corresponds to approximately 1 PE out of 64, the GDPS can assume a user arrival rate of up to 1 user per second for m = 10, ρ = 0.5, and n = 64. For higher m, ρ, or n, the scheduling invocation rate needs to be even lower. The PSS, on the other hand, can manage much more than 64 PEs or manage more than 10 new users per second. All four figures indicate that the PSS can manage 100 times higher user arrival rates than the GDPS for equivalent overheads and n = 64.

Note that n PEs will execute n/k waveforms at most. High λ values then do not make sense for small clusters. We may, however, extend the scope of λ embracing also the dynamic switches in transmission modes that require reconfigurations and rescheduling.

4. CONCLUSIONS

This paper has analyzed the complexity of two popular scheduling concepts for their applicability in SDR clouds. The results confirm the common scalability problem of global schedulers [2]. Moreover, whereas global scheduling works with multicores, it is very inefficient for multiprocessors because of the costly task migrations. In the SDR cloud this may considerably limit the definition of clusters.

The PSS, on the other hand, scales well with the number of PEs. The limiting factor here is the user arrival rate, which may impede the definition of large clusters serving large geographical areas with high user mobility.

We should mention that some RATs, such as WiFi or WiMAX, send packets in bursts based on the channel availability. The signal processing is still data driven, but asynchronous. This may have an effect on the applicability of the proposed PSS scheme and requires a thorough analysis, which is beyond this paper's scope. Modern transceivers may, furthermore, allow for frequent changes in the transmission and reception modes. If these changes involve the reconfiguration of processing blocks, the PSS would need to find robust resource allocations or remap the corresponding waveform (tasks). The scheduling would then be static only for the duration of a configuration rather than the entire user session. A hybrid approach between static and dynamic scheduling may therefore be explored.

ACKNOWLEDGMENT

This work was supported by Spanish Government (MINECO) under project TEC2011-29126-C03-02.

5. REFERENCES

[1] V. Marojevic, X. Revés, A. Gelonch, "A computing resource management framework for software-defined radios," *IEEE Trans. Comput.*, vol. 57, no. 10, pp. 1399-1412, Oct. 2008.

[2] B. B. Brandenburg, J. M. Calandrino, J. H. Anderson, "On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study," *Proc. 2008 Real-Time System Symposium*, pp. 157-169, 2008.

[3] Z. Zhu, et al., "Virtual base station pool: towards a wireless network cloud for radio access networks," *Proc. 8th ACM Int. Conf. Comp. Frontiers (CF'11)*, 3-5 May, 2011, Ischia, Italy.

[4] V. Marojevic, "Computing resource management in software-defined and cognitive radios," Ph.D. Dissertation, Universitat Politècica de Catalunya (UPC), Barcelona, July 2009. Available at http://flexnets.upc.edu/trac/wiki/Publications

[5] T. L. Casavant, J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141-154, Feb. 1988.

[6] D. H. Woo; H.-H.S. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era," *IEEE Computer*, vol. 41, iss 12, pp. 24-31, Dec. 2008.

[7] M. A. Dellinger, "An experimental evaluation of the scalability of real-time scheduling algorithms on large-scale multicore platforms," MSc Thesis, Virginia Tech, 2011.

[8] S. H. Bokhari, "On the mapping problem," *IEEE Trans. Comput.*, vol. C-30, no. 3, pp. 207-214, March 1981.

[9] X.M. Zhu, P.Z. Lu, "Multi-dimensional scheduling for real-time tasks on heterogeneous clusters," *J. of Computer Science and Technology*, vol. 24, iss. 3, pp. 434-446, May 2009.

[10] O. Arnold, G. Fettweiss, "On the impact of dynamic task scheduling in heterogeneous MPSoCs," *Proc. 2011 Int. Conf. Embedded Computer Systems (SAMOS)*, pp. 17-24, 2011.

[11] I. Gomez, V. Marojevic, A. Gelonch, "Resource management for software-defined radio clouds," *IEEE MICRO*, vol. 32, iss. 1, pp. 44-53, Jan/Feb 2012.