Towards Implementation of Virtual-Clustered Multiprocessor Scheduling in Linux

Syed Md Jakaria Abdullah, Nima Moghaddami Khalilzad, Moris Behnam, Thomas Nolte
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
nima.m.khalilzad@mdh.se

***Abstract*—Cluster based multiprocessor scheduling can be seen as a hybrid approach combining benefits of both partitioned and global scheduling. Virtual clustering further enhances it by providing dynamic cluster resource allocation and applying hierarchical scheduling techniques. Over the years, the study of virtual cluster scheduling has been limited to theoretical analysis. In this paper, we present our initial ideas on implementing virtual cluster scheduling in Linux. The purpose of this implementation is twofold: (i) we would like to demonstrate the feasibility of its implementation in an operating system, without modifying the kernel source code, (ii) we present practical insights on the overhead of implementing this framework.**

I. Introduction

In recent years, we have witnessed a major paradigm shift in the computing platform design. Single-core chip designs suffer from many physical limitations such as excessive energy consumption, chip overheating, memory size and memory access speed which can be reduced by placing multiple processing cores that share some levels of cache memories on the same chip. As a result, the hardware vendors now prefer to increase the number of processors available in a single chip.

The study of real-time scheduling in multiprocessors dates back to early 70s [1], even before the appearance of the actual hardware. Recent advancement of the processing platform architectures led to a regain of interest for the multiprocessor real-time scheduling theory during the last decade. The two most popular scheduling approaches in multiprocessors are global and partitioned scheduling [2]. In partitioned scheduling a task set is partitioned into disjoint partitions and each of the partition is independently scheduled in its assigned processor. In contrast, global scheduling uses only one global scheduler to schedule all the tasks in all the available processors. However, partitioned scheduling suffers from inherent algorithmic complexity of partitioning, and global scheduling is not scalable due to scheduler overhead [2].

Recently, a number of hybrid approaches combining the partitioned and global scheduling methods have been proposed, e.g, semi-partitioned [3] and cluster based scheduling [4]. The main idea of cluster based scheduling is to divide m processors into $\lceil \frac{m}{c} \rceil$ sets of c processors each. Both partitioned and global scheduling can be seen as extreme cases of clustering with $c = 1$ and $c = m$, respectively. Clustering reduces the partitioning problem by reducing the number of partitions, and distributes the overhead of the global scheduler into per-cluster schedulers. The notion of physical cluster of processors has been enhanced by Shin et al. [5] using Virtual Clusters (VCs). VCs are dynamically mapped into a set of available processors via a resource interface. Therefore, virtual clustered scheduling appears to be more flexible than the original physical clustering [4]. Although Easwaran et al. [6] provided a complete hierarchical scheduling framework for implementing virtual clustering, there is no experimental implementation of it to the best of our knowledge.

In this paper, we present our work towards the implementation of a Virtual-Clustered Hierarchical Scheduling Framework (VC-HSF) in Linux without modifying the base Linux kernel. This work includes our design of the framework and related implementation challenges. To avoid modification of the Linux kernel, we intend to use the ExSched framework [7].

II. Related Work

In this section, we present related work in clustered and hierarchical multiprocessor scheduling.

A. Clustered Multiprocessor Scheduling

In $LITMUS^{RT}$, the Clustered EDF (C-EDF) algorithm is implemented to compare its performance with respect to other multiprocessor algorithms [8]. The main idea of C-EDF is to group multiple processors that share a cache (either L2 or L3) into one cluster and to assign tasks to the cluster off-line. Each cluster has a separate runqueue and during run-time it uses a global scheduling algorithm within the cluster. Tasks can only migrate between the processors of their own cluster and different clusters do not share processors. Lelli et al. [9] also implemented C-EDF in a multiprocessor extension of the customized Linux scheduling class `SCHED_DEADLINE` [10]. Both of these implementations of C-EDF are examples of physical clustered scheduling for multiprocessors and they rely on patch based modification of the Linux kernel.

However, the focus of our work is on virtual clustered scheduling which differs from C-EDF in several aspects. Firstly, unlike physical clusters, VCs can share processors. Secondly, instead of assigning processors to a cluster off-line, virtual clustering can assign them on-line using global scheduling. Finally, the task migration is not limited to a set of processors (like clustered processors of C-EDF), as processors assigned to a cluster can change dynamically.

B. Hierarchical Multiprocessor Scheduling

Two-level hierarchical scheduling [11] that has been introduced for uniprocessor platforms provides a temporal isolation

mechanism for different components (subsystems). This is increasingly becoming important due to the component-based nature of systems' software development. There are several models to abstract the resource requirements of components such as the bounded delay model [12] and the periodic resource model [13]. These resource models are extended to hierarchical multiprocessor scheduling such as the Multiprocessor Periodic Resource (MPR) model [5] and the Bounded-Delay Multipartition (BDM) [14] model. Both the MPR and BDM models are proposed as part of a hierarchical scheduling framework which comprises schedulabilty analysis, resource interface generation and run-time allocation. As shown in the BDM, in the original MPR it is assumed that the servers on different processors are synchronized. However, this assumption is relaxed in [15]. To the best of our knowledge none of these frameworks is actually implemented in a multiprocessor platform.

Different implementation schemes for hierarchical scheduling for multiprocessors are analyzed in [16]. Checconi et al. [17] have implemented a two-level hierarchical scheduling for multiprocessors in Linux. Their implementation exploits the hierarchical resource management and task group scheduling support in Linux via $cgroups$ and $throttling$ mechanisms. However, their implementation of hierarchical scheduling requires multiple global schedulers and each of the subsystem can access all processors. Additionally, this implementation is patch based, thus it requires modification of the base Linux kernel.

Hierarchical compositional scheduling has been realized in [18] and [19] through virtualization. However, our work is different from these papers in two aspects. Firstly, we intend to implement the complete hierarchy of schedulers within a single operating system. Secondly, none of the previous works [18], [19] addressed the MPR interface as the resource interface model.

III. Background

A. System Model

In this paper, we consider a simple sporadic task model $\tau_i^j(T_i^j, E_i^j, D_i^j)$ where T_i^j is the minimum inter-arrival time, E_i^j is the worst-case execution time requirement, and D_i^j is the relative deadline $(0 < E_i^j \leq D_i^j \leq T_i^j)$. The superscript in the task notation represents the cluster id. The set of all tasks is denoted by $\Gamma(\Gamma = \tau_i^j | \forall i = 1, .., n)$ where n is the number of tasks. We assume that all tasks are independent of each other and each job of τ_i^j must be supplied with C_i^j units of the processor capacity non-concurrently within D_i^j time units after its release.

A time-driven periodic server [20] is defined as $PS_i(P_i, Q_i)$, where P_i is the server period, and Q_i is the server budget which represents the number of CPU time units that has to be provided by the server every P_i time units. The periodic servers idle their budget if there is no active task running inside the server.

B. Virtual Cluster Scheduling

The virtual cluster scheduling framework [5] is a generalization of physical clustering with a new feature of sharing processors between different clusters. Unlike physical clusters, where processors are dedicated to a cluster off-line, VC-HSF allows allocation of physical processors to the clusters during run-time. This dynamic allocation scheme requires an interface to capture the execution and concurrency requirements within a cluster to use hierarchical scheduling techniques. The interface proposed by Shin et al. [5] which is known as the MPR model is:

Definition 1. *The MPR model $\mu =< \Pi, \theta, m' >$ where $\theta \leq \Pi$ specifies a unit capacity, identical multiprocessor platform with at most m' processors can collectively supply θ units of execution resource in every Π time units. At any time instance at most m' processors are allocated concurrently to μ where θ/Π denotes the bandwidth of model μ [5].*

In VC-HSF, a sporadic task set Γ is assigned to a set of clusters and for each cluster C_i an MPR interface μ_i is generated using the schedulability analysis method presented by Shin et al. [5]. Then each of these interfaces is transformed into a set of implicit deadline periodic servers for inter cluster scheduling. The result of this transformation is one to m' periodic servers with a common period equal to Π. All of the periodic servers associated with each cluster can collectively consume θ time units which is called the total budget. The total budget is reduced when any of the servers is running. One example for mapping the cluster interface to periodic servers is the method proposed by Easwaran et al. [6] which works as follows. Given an MPR interface $\mu_j =< \Pi_j, \theta_j, m'_j >$ for cluster C_j, it creates a set of implicit deadline periodic servers $PS_1^j, \ldots, PS_{m'_j}^j$, where,

$$PS_1^j = PS_2^j = \ldots = PS_{m'_j-1}^j = (\Pi_j, \Pi_j) \quad (1)$$

$$PS_{m'_j}^j = (\theta_i - (m'_j - 1).\Pi_j, \Pi_j). \quad (2)$$

The servers $PS_1^j, \ldots, PS_{m'_{j-1}}$ are full budget servers, while $PS_{m'_j}$ is a partial budget server.

Once all the interfaces are transformed into the periodic servers, VC-HSF uses hierarchical scheduling to schedule servers and tasks. There are two levels of scheduling described in VC-HSF, namely inter-cluster scheduling and intra-cluster scheduling. Here the inter-cluster scheduler refers to the global scheduler of the hierarchical scheduling while the notion of intra-cluster scheduler is similar to the local scheduler in hierarchical scheduling.

In hierarchical scheduling, the global scheduler schedules the servers representing the subsystem. The same is true for the inter-cluster scheduler of VC-HSF except that each cluster can have up to m' active servers. All the servers from all the clusters are queued according to the global scheduling policy. However, tasks are not assigned to any particular server, rather these only belong to a specific cluster. The intra-cluster executes tasks of the cluster by consuming the budgets of its scheduled servers. Unlike regular hierarchical scheduling, the local or intra-cluster scheduler also has to use a multiprocessor global scheduling algorithm as there can be multiple active servers of a cluster. As a result, VC-HSF can be described as

global scheduling in two level.

C. ExSched

ExSched [7] is a scheduling framework that can be used to implement custom schedulers as plug-ins for different operating systems without changing the kernel of the operating system. It consists of three major components: a core kernel module, a set of scheduler plug-ins and a library for the user space programs.

The key component of the ExSched framework is its core module. It is a character-device module which can be loaded into kernels which support loadable modules. The core module is accessed by the user space programs through I/O system calls such as `ioctl()`. To determine the scheduling decisions for the user program, the core module invokes a set of callback functions implemented by the specific scheduler plug-in. To develop a plug-in in ExSched, the designer has to implement these callback functions that will be used by the core module. Finally, the core module implements custom scheduling decisions from the plug-ins via original scheduling primitives of the host operating system. For example, in Linux, the ExSched core uses `SCHED_FIFO` policy of the real-time scheduling class `rt_sched_class` to implement the scheduling decisions from the plug-ins. It uses scheduling functions provided by the Linux kernel such as `schedule`, `sched_setscheduler` and `set_cpus_allowed_ptr` to implement real-time scheduling. In case of multiprocessors, task migration is done in two ways. The `migrate_task(task,cpu)` function of the core can migrate a task running in the thread context by simply calling the `set_cpus_allowed_ptr`. However, in case of task running in the interrupt context, the core module has to create a high priority real-time kernel thread to migrate the task.

IV. Design of VC-HSF

In this section we describe different components of the design of VC-HSF.

A. Inter-cluster Scheduling

The single inter-cluster scheduler of VC-HSF is similar to the global scheduler of the conventional hierarchical scheduling approach. To implement global scheduling, the inter-cluster scheduler needs to use two queues to manage servers. These are a global release queue for servers waiting to be released and a global ready queue for all the ready servers. Different inter-cluster scheduling algorithms can be implemented by manipulating the management of these queues.

Each of the servers in a cluster needs its own server descriptor defined by the structure vc_server_t. There are two major differences in server descriptors of VC-HSF compared to the server descriptor of any other hierarchical scheduling. Firstly, as tasks are not linked to a server in VC-HSF, there is no relation between server descriptors with the queues of tasks. Instead, the server descriptor only needs a reference to the descriptor of its cluster. Secondly in VC-HSF, servers can migrate from one processor to another, so the server descriptor needs to keep information about the processor where it is running. In addition, similar to other server descriptors, vc_server_t contains general server parameters such as the server period and deadline.

The functions of the inter-cluster scheduler can be summarized as following: whenever it is invoked, either the total budget is expired or it gets replenished via activation. If a server is released as the highest priority one in the server ready queue, then the scheduler first has to check for an idle processor where it can run that server. If there is no idle processor then it has to check all the busy processors to find if there is a server running with a lower priority that it can preempt. In case of server preemption, it should manage the remaining budget of the preempted server's cluster and it should insert the server into a proper place in the server ready queue. If both attempts fail, the server remains in the ready queue. When the total budget of a cluster expires, the inter-cluster scheduler inserts the descriptor of its associated servers in the server release queue. If the server ready queue is not empty then the scheduler picks the highest priority ready server to run on the idle processor. It invokes the intra-cluster scheduler whenever a server of that cluster either gets a new budget, gets preepmted or depletes its budget. The inter-cluster scheduler also migrates the server by changing the processor id of the server when a server is scheduled on a different processor, than its previous run.

In [6], the McNaughton's algorithm is proposed for inter-cluster scheduling. However, as this off-line algorithm requires the same period for all the clusters, we want to use another general global scheduling algorithm such as the global EDF (gEDF) algorithm for inter-cluster scheduling.

B. Intra-cluster Scheduling

The VC-HSF needs one intra-cluster (local) scheduler per cluster instead of one local scheduler per server of the conventional hierarchical scheduling. Each cluster needs its own release queue and ready queue for managing its tasks. All the servers in a single cluster share task queues of that cluster. However, the intra-cluster scheduler employs a multiprocessor global scheduling algorithm to schedule tasks into processors as there can be more than one processor available for the cluster. The inter-cluster scheduler works in the following way: it is invoked by the inter-cluster scheduler whenever one of its server's total budget is replenished or it is depleted or an associated server gets preempted by the other servers of a different cluster. It is also invoked when a new task job of its cluster becomes ready or completes its execution. If a newly released job of the cluster is the highest priority one in the ready queue then the scheduler first has to check for an idle server within its own cluster to run that job using its budget. If there is no idle server, then it has to check all the busy servers of its cluster to find if there is a task job running with lower priority that it can preempt. If there is no task to preempt, the new job should remain in the ready queue of its cluster. It inserts released and preempted task jobs into the task ready queue of the cluster. The completed jobs are also inserted into the cluster wise release queue for subsequent release. The intra-cluster scheduler migrates a task to a new processor whenever

the server running that task is migrated by the inter-cluster scheduler.

Each cluster requires a cluster descriptor vc_cluster_t. The descriptor contains a reference to the task queues of the cluster and also a list of its active running servers. The original task descriptor provided by the ExSched also needs to be extended such that vc_task_t assigns each task to a cluster. As the processors are assigned to the clusters dynamically, and a single cluster can have more than one processor, the cluster task queues of clusters are implemented globally. A brief overview of the framework is illustrated in Figure 1.

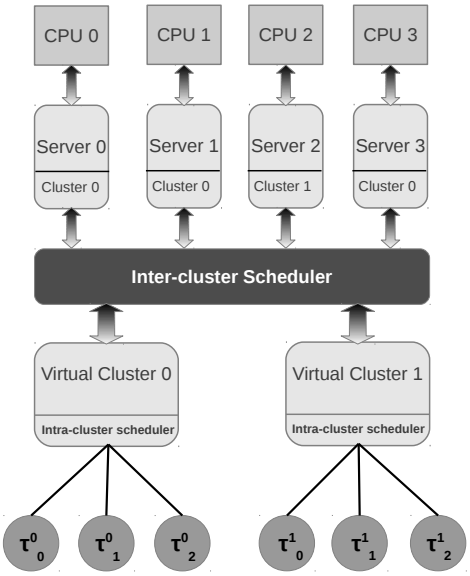

Figure 1. Design overview of VC-HSF

V. Future Work

Currently we are working on the following three aspects of the virtual clustered hierarchical scheduling framework implementation using ExSched [7].

- We are investigating the best way to implement inter-cluster and intra-cluster scheduling using plug-in functions of ExSched. For this work, we are reusing some available functionalities of the ExSched's uniprocessor hierarchical scheduling plug-in.
- We are investigating how to efficiently implement both server and task migration mechanisms using ExSched.
- In the virtual cluster hierarchical scheduling framework, the inter-cluster scheduler needs to check all busy processors to find a server to preempt. Similarly, the intra-cluster scheduler needs to check all running servers in its cluster to find a task to preempt. We are investigating how to implement this searches efficiently using ExSched.

Acknowledgment

The authors would like to thank Mikael Åsberg for helping with knowledge related to ExSched.

References

[1] C. L. Liu, "Scheduling Algorithms for Multiprocessors in a Hard Real-Time Environment," *JPL Space Programs Summary 37-60*, vol. II, pp. 28–31, 1969.

[2] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, pp. 1–44, 2011.

[3] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, August 2006, pp. 322–334.

[4] J. Calandrino, J. Anderson, and D. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," in *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, April 2007, pp. 247–258.

[5] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *20th Euromicro Conference on Real-Time Systems (ECRTS'08)*. IEEE Computer Society, April 2008, pp. 181–190.

[6] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25–59, 2009.

[7] M. Åsberg, T. Nolte, S. Kato, and R. Rajkumar, "Exsched: An external cpu scheduler framework for real-time systems," in *18th IEEE conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'12)*, August 2012, pp. 240–249.

[8] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers," in *31st IEEE International Real-Time Systems Symposium (RTSS'10)*, December 2010, pp. 14–24.

[9] J. Lelli, G. Lipari, D. Faggioli, and T. Cucinotta, "An efficient and scalable implementation of global edf in linux," in *7th annual workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT'11)*, July 2011.

[10] D. Faggioli, M. Trimarchi, F. Checconi, and S. Claudio, "An edf scheduling class for the linux kernel," in *11th Real-Time Workshop (RTLW'09)*, October 2009.

[11] Z. Deng and J. W. S. Liu, "Scheduling real-time applications in an open environment," in *18th IEEE Real-Time Systems Symposium (RTSS'97)*, December 1997, pp. 308–319.

[12] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *7th Real-Time Technology and Applications Symposium (RTAS'01)*, May 2001, pp. 75–84.

[13] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *24th IEEE Real-Time Systems Symposium (RTSS'03)*, December 2003, pp. 2–13.

[14] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *31st IEEE Real-Time Systems Symposium (RTSS'2010)*, December 2010, pp. 249–258.

[15] N. M. Khalilzad, M. Behnam, and T. Nolte, "Exact and approximate supply bound function for multiprocessor periodic resource model: Unsynchronized servers," in *5th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'12)*, December 2012, pp. 1–8.

[16] M. Åsberg, T. Nolte, and S. Kato, "Towards hierarchical scheduling in linux/multi-core platform," in *15th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'10)*, September 2010, pp. 1–4.

[17] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical multiprocessor CPU reservations for linux kernel," in *5th annual workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT'09)*, July 2009.

[18] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin, "Implementation of compositional scheduling framework on virtualization," *SIGBED Review*, vol. 8, no. 1, pp. 30–37, March 2011.

[19] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing compositional scheduling through virtualization," in *18th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'12)*, April 2012, pp. 13–22.

[20] R. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *26th IEEE Real-Time Systems Symposium (RTSS'05)*, December 2005, pp. 10–398.