Load-Balancing for a Real-Time System Based on Asymmetric Multi-Processing[*]

Éric PIEL Philippe MARQUET Julien SOULA Jean-Luc DEKEYSER
www.lifl.fr/west/artis
Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille
France

Abstract

ARTiS is a project that aims at enhancing the Linux kernel with better real-time properties. It allows to retain the flexibility and ease of development of a normal application for the real-time applications while keeping the whole power of SMP (Symmetric Multi-Processors) systems for their execution.

Based on the introduction of an asymmetry between the processors, distinguishing real-time and non real-time processors, the system can insure low interrupt latencies to real-time tasks. Furthermore, every processor can execute all the tasks, excepted when they request real-time endangering functions. In this case the task is moved before continuing to be executed. A first version of ARTiS has demonstrated this is technically possible. Unfortunately, the original load-balancing mechanism of Linux is not aware of this enhanced design.

We have studied all the types of migration possible between the combinations of a real-time specialized processor and a general one. From the deducted requirements, we have specified special mechanisms and policies taking into account both performances and real-time specificities. We are currently working on implementing those particular load-balancing functions within the ARTiS system.

1 Real-Time and Load-Balancing on SMP

There exists two types of approach to obtain real-time properties from the Linux kernel. One consists in running the RT tasks in a special designed kernel running in parallel, this is what does RTAI [2]. The drawback is that the programming model and configuration methods are different from the usual one: Linux tasks are not real-time tasks and real-time activities can not benefit of the Linux services. The second approach relies on the shielded processors or asymmetric multi-processing principle. On a multi-processor machine, the processors are specialized to real-time or not. Concurrent Computer Corporation RedHawk Linux variant [1] and SGI REACT IRIX variant [5] follow this principle. However, since only RT tasks are allowed to run on shielded CPUs, if those tasks are not consuming all the available power then there is free CPU time which is lost. ARTiS extends this second approach by also allowing normal tasks to be executed on those processors as long as they are not endangering the real-time properties.

ARTiS insures a possible processor preemption when the system has to schedule a real-time process [4]. The main principle is to distinguish two kinds of CPUs in the multi-processor system: specialized CPUs, a part oriented toward real-time (the so-called RT CPUs) and another part serving all the other tasks (the so-called non real-time CPUs, NRT CPUs). Two types of RT tasks are characterized, the RT0 which have the highest priority and are binded to a processor and the RT1+ which have lower priority and may migrate between processors. Every task is allowed to run on an RT CPU but tasks which are not RT0 will not be allowed to perform real-time endangering functions on this particular CPU. We have implemented ARTiS in the 2.6 Linux kernel and tested it on x86 and IA-64 plat-

[*]This work is partially supported by the ITEA project 01010, HYADES

forms. The implementation of the ARTiS system relies on the fact that any call to `preemt_disable()` or `local_irq_disable()` from a task without the highest RT priority leads systematically to the migration of this task to an NRT CPU. Thus the RT CPUs remain able to face to real-time activities without long latency.

Nevertheless, the system has to insure a migration mechanism: a NRT task must leave a RT CPU if it jeopardizes the real-time response time on this RT CPU. Furthermore, to maximize the utilization of the whole of the CPU, those NRT tasks may have to come back on the RT CPU latter. The standard Linux kernel already provides a migration mechanism. Unfortunately, this standard mechanism is not sufficient because it is not aware of the specialization of the different processors and also because it holds in the same time locks of two different CPUs (possibly one being a RT CPU) to insure the task migration: the RT CPU may have to wait after the completion of an operation on a NRT CPU which is unacceptable. In order to address those issues we have to design a specialized load-balancing mechanism.

Usually, the load-balancing mechanism aim is to move the running tasks across the CPUs in order to insure that no CPU is idle while some tasks are waiting to be scheduled on other CPUs. It should minimize the total running time by a set of tasks. The characteristics of a load-balancing can be enumerated as follow:

- information update policy: how to renew statistics about the entire system,
- trigger policy: how to decide it is time to redistribute the tasks,
- selection policy: method to select imbalanced nodes,
- local designation policy: method to select the tasks that will move,
- pairing policy: method to select the destination node for a given task.

The trigger policy can be either of type "pull" – the low loaded CPUs initiate the load-balancing and pull the tasks from another CPU– or "push" –overloaded CPUs initiate the load-balancing in order to push some of their tasks– or a mix of both.

2 ARTiS Migrations

In order to specify a more advanced load-balancing mechanism it is necessary to distinguish the different types of migration according to the specialization of the CPUs involved: RT and NRT CPUs. There may be several mechanisms associated to a given kind of migration in order to fit all the scenarii involving this migration.

NRT CPU to NRT CPU This migration is used for load-balancing between non real-time CPUs. As no RT CPU is involved there is not particular requirement to take care. The original Linux mechanism [3] can be kept for this kind of load-balancing.

RT CPU to NRT CPU This type of migration is mainly called when a task on a RT CPU tries to disable the preemption (endangering the RT properties). This is the core mechanism of ARTiS and it is already implemented. There is also a load-balancing mechanism related to this migration. When a NRT CPU has less load than a RT CPU, some of the NRT tasks should be moved to the NRT CPU. RT tasks should not be moved as there is better response time on the RT CPUs. In practice, most of the tasks trigger preemption disabling code often enough so that this load-balancing is usually not needed. Still, it is necessary to handle this case in order to guaranty the best use of all the CPUs in every configuration (for instance with tasks doing only computational work). In this kind of migration it is important that the RT CPU does not take a lock shared with the NRT CPU.

NRT CPU to RT CPU This migration is employed in two different contexts. First, it is used to move back as soon as possible to a RT CPU the RT tasks which have reenabled the preemption. In the ARTiS model, the highest priority RT tasks are always on a RT CPU but other RT tasks may migrate to a NRT CPU. Therefore, to provide the smallest latencies to them it is necessary to bring them back to any RT CPU as soon as they are allowed to. However, as the migration process costs some time it is also important not to move back a task that may need soon to migrate again. Although it is obviously impossible to exactly know in advance the future behavior of a given task, the local designation part of the load-balancing algorithm has to approximately predict the next time of migration.

Second, this migration is required to load-balance the NRT tasks if a RT CPU has free time available. Even if the most important point is to keep the RT properties on the RT CPUs, this load-balancing represent a major advantage of ARTiS and so it must not be neglected.

Both moves between the CPUs implies the modification of the runqueue of each processor. If the NRT CPU locks the runqueue of the RT CPU (which is the standard mechanism) then the RT properties of this latter cannot be guaranteed. It is therefore necessary to find a transfer mechanism which is lock-free at least on the RT CPU side.

RT CPU to RT CPU The migration between two RT CPUs is solely used to balance their load. The algorithm can be very similar to the NRT to NRT algorithm with the exception that it has to avoid locks between two CPUs that will possibly lead to a jitter on the expected latencies.

3 Migration Implementation

Starting from the specific requirements described above, we have defined the algorithms and implementations of the dedicated load-balancing mechanisms in ARTiS.

Lock-free queues One of the main change which is required from the original load-balancing mechanism is the removal of inter-CPU locks. In order to be able to insure the RT properties of the RT CPUs, there should not be locks that can be taken both by NRT and RT CPUs. If a RT CPU tries to take a lock already taken by a NRT CPU it will have to wait after it. To avoid this particular sequence we decided to avoid taking shared locks. The only shared lock is on the runqueue which has to be modified from the other CPU when inserting a migrating task. This is why this mechanism has to be changed to an indirect one. Instead of removing the task from a runqueue and adding it to another runqueue, we insert it to a special FIFO connecting the two CPUs. The task is dequeued later and asynchronously by the second CPU. Because this FIFO has only one writer and one reader, it is possible to access it without lock, as described in [6]. We are not expecting machines with more than 32 CPUs, so even if there are two queues per couple of CPUs, the size of these data structure should never be excessive.

Trigger policy By default, the Linux kernel [3] uses a "pull" policy for the load-balancing. However, with the FIFO mechanism which is required to avoid the locks, this policy has a more complex implementation (leading to longer delays) than the "push" policy. The use of a queue with this second policy is straightforward. Consequently, for the new load-balancing functions, we invert the default policy. A CPU will look for the less busy CPU and then select tasks to send from its own runqueue to the second CPU. In order to lower the latency between the moment a task is inserted and the moment the second CPU reads the FIFO, a signal (an IPI - Inter-Processor Interrupt) is sent, warning about a new task in the queue. In addition, concerning the RT tasks on a NRT CPU, the time spent on this processor must be minimized so this special load-balancing function is triggered more often than the other functions, at every scheduling (typically every millisecond).

Local designation criteria The designation mechanism of the load-balancing which is charged to decide which task is better to migrate has to be adapted to each type of load-balancing. For the symmetric load-balancing (NRT to NRT and RT to RT) the original criteria can be kept as the requirements are the same than in a normal configuration. For the RT to NRT CPUs the only additional criterion is to avoid RT tasks (it is better to let them on the RT CPU). Concerning the NRT to RT CPUs migrations, there are two kinds of load-balancing. One is to move back as soon as possible the RT tasks and a second is to use the free cycles of the RT CPUs. Both functions have to predict if a task will soon have to migrate again to a NRT CPU because it requires preemption disabled. We propose to estimate the likelihood of another migration by the frequency of the previous migration attempts: we suppose a task that has not disabled preemption for a long time is less likely to disable it in a close future. The implementation of this prediction can be done by saving the time of the last two migration attempts of each task. From those times, we obtain the time weighted mean of the time elapsed between two attempts. Then the selection criterion do not move tasks if the time since their last migration attempt multiplied by a constant K is smaller than the mean. K has to be specified later following the results of experiments, it is expected to be between 2 and 100.

Pairing policy The original kernel mechanism compares the load of processors simply by using the number of tasks ready to run on each processor (runqueue length). Although this method is sufficient for a system executing nearly exclusively NRT tasks, it gives unexpected results if real-time tasks consume a significant amount of time because they do not share time with lower priority tasks. Therefore, the pairing policy has to be enhanced to take into account, in addition to the runqueue length, the time consumed by RT tasks. In the pairing procedure, we ponderate the runqueue length by $\frac{1}{(1-RT)}$, where RT is the ratio of CPU time used by the RT tasks.

4 Conclusion

In this paper, we have described the specific migration types and their requirements for a real-time system based on an asymmetry of the processors. We distinguish load-balancing strategies according to RT and NRT tasks. We have also pointed out that inter-CPU locks must be avoided. We have then presented the techniques on which our solution are based. Using lock-free FIFOs and an trigger policy of type “push”, it is possible to implement lock-free load-balancing functions. The real-time aware behavior is achieved via the introduction of new running statistics (elapsed time between two migration attempts, CPU time consumed by RT tasks) and the specialization of the functions according to the migration type.

For now, a first version of ARTiS has been developed. It integrates the migration of the tasks which disable preemption from an RT to NRT CPU via lock-free queues. The normal load-balancing from the RT to NRT CPUs has been deactivated. However, this standard load-balancing from NRT to RT CPUs is still active (otherwise, tasks would never come back to the RT CPUs) and leads to high latencies. In the same time, measurements have been done on a simulated ARTiS system with a static configuration (tasks attached by hand to the processors). A huge improvement of the latencies could be noticed over a normal kernel: all the latencies of a RT0 task were under 30μs [4].

The future work will first consist of the full implementation of the proposed algorithms. The existing version of ARTiS will be the starting point for a version including the implementation of the load-balancing functions. Then the major part of the work will be spent on designing measurement and verification procedures. They will first be used to check and fine-tune the implemented code and later to perform comparisons with other kernels and configurations.

References

[1] Steve Brosky and Steve Rotolo. Shielded processors: Guaranteeing sub-millisecond response in standard Linux. In *Workshop on Parallel and Distributed Real-Time Systems, WPDRTS'03*, Nice, France, April 2003.

[2] Pierre Cloutier, Paolo Montegazza, Steve Papacharalambous, Ian Soanes, Stuart Hughes, and Karim Yaghmour. DIAPM-RTAI position paper. In *Second Real Time Linux Workshop*, Orlando, FL, November 2000.

[3] Robert Love. *Linux Kernel Development*. Sams Publishing, August 2003.

[4] Philippe Marquet, Julien Soula, Éric Piel, and Jean-Luc Dekeyser. An asymmetric model for real-time and load-balancing on Linux SMP. Research Report 2004-04, Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille, France, April 2004.

[5] Sillicon Graphics, Inc. REACT: Real-time in IRIX. Technical report, Sillicon Graphics, Inc., Mountain View, CA, 1997.

[6] John D. Valois. Implementing lock-free queues. In *Proceedings of the Seventh International Conference on Parallel and Distributed Computing Systems*, Las Vegas, NV, October 1994.