An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers*

Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

As multicore platforms become ever larger, overhead-related factors play a greater role in determining which real-time scheduling algorithms are preferable. In this paper, such factors are investigated through an empirical comparison of global, partitioned, and clustered EDF scheduling algorithms on a 24-core Intel system. On this platform, global EDF proved to be a non-viable choice for hard real-time systems, while clusters of size six practically approximated global approaches. For soft real-time systems, clustered EDF scheduling algorithms proved to be particularly effective. This study suggests that future global scheduling research should focus on small-to-medium multicore platforms rather than large platforms.

1 Introduction

In future multicore chip designs, per-chip core counts are expected to increase significantly. Evidence of this can be seen in recent announcements by major chip manufacturers. For example, Intel recently presented plans for a many-core platform that features more than 50 cores per chip [19]. Even now, rather large, complex platforms exist. Examples include AMD's "Magny-Cours" Opteron processors, with 12 cores per chip, and Intel's "Beckton" Xeon processors, with eight multi-threaded cores per chip.

In this paper, we consider the question on how to best schedule *implicit deadlines sporadic tasks* on such large platforms where timing constraints may be either hard or soft. The soft real-time constraint considered in this paper is that *deadline tardiness* be bounded.

Multiprocessor scheduling approaches. Two basic approaches exist for scheduling real-time tasks on multiprocessor platforms. In the *partitioned* approach, each task is statically assigned to a single processor and migration is not allowed; in the *global* approach, tasks can freely migrate and execute on any processor. On large multiprocessor platforms, such as those mentioned above, both approaches suffer drawbacks that limit achievable processor utilizations. On a platform with m cores, partitioning algorithms require that a bin-packing-like problem be solved to assign tasks to processors. Because of such bin-packing connections, restrictive caps on total utilization are generally required to ensure timing constraints, for *both* hard real-time (HRT) and soft real-time (SRT) systems. Most global approaches require similar caps in the HRT case, but in the SRT case, many global algorithms ensure bounded deadline tardiness, as long as total utilization is at most m [16]. However, due to contention for the global run queue and non-negligible migration overheads among processors, global approaches can have high overheads in practice.

As a compromise that aims to alleviate limitations of partitioned and global algorithms, *clustered* scheduling has been proposed [3, 13]. This approach exploits the grouping of cores around different levels of shared caches. The platform is partitioned into *clusters* of cores that share a cache and tasks are statically assigned to clusters (like in partitioning), but are globally scheduled within each cluster.

Cluster-size guidelines have been given in [13], but these guidelines refer to SRT systems only and are based on measurements taken using an architecture simulator. Indeed, when implementing clustered algorithms on real systems, many unanswered questions exist. What is the best shared cache level to use for clustering? Will the chosen cluster size perform equally well for HRT and SRT systems? How does the impact of various preemption- and migration-related overheads compare to scheduling overheads? In this paper, we answer these questions by empirically comparing implementations of *global*, *partitioned*, and *clustered earliest-deadline-first* (G-EDF, P-EDF, C-EDF) scheduling algorithms on a large multiprocessor platform.

Prior work. Fundamental questions concerning the viability of supporting sporadic real-time workloads on symmetric multi-processor (SMP) and multicore platforms *under consideration of real-world overheads* have been tackled at UNC in several different studies. To facilitate this line of research, a real-time Linux extension called LITMUS$^{\mathrm{RT}}$ (**LI**nux **T**estbed for **MU**ltiprocessor **S**cheduling in **R**eal-**T**ime systems) was developed. LITMUS$^{\mathrm{RT}}$ allows different (multiprocessor) scheduling algorithms to be implemented as plugin components [14, 18]. To the best of our

*Work supported by AT&T and IBM Corps.; NSF grants CNS 0834270 and CNS 0834132; ARO grant W911NF-09-1-0535; and AFOSR grant FA 9550-09-1-0549.

knowledge, LITMUS$^{\mathrm{RT}}$ is the only (published) real-time OS where both global and clustered real-time schedulers are supported.

In the first study [14], Calandrino *et al.* evaluated five well-known multiprocessor real-time scheduling algorithms on a four-processor (non-multicore) 32-bit 2.7 GHz Intel Xeon SMP platform. On this small SMP platform, with relatively large *private* L2 caches, each tested algorithm proved to be the preferred choice in some of the tested scenarios. In particular, global algorithms outperformed partitioned algorithms in supporting SRT workloads.

In a second study [11], Brandenburg *et al.* analyzed the *scalability* of several global, partitioned, and clustered algorithms including the EDF variants mentioned above. This evaluation was conducted on a much larger and slower multicore platform: a SUN Niagara with a small *single* shared L2 cache and 32 logical processors, each with an effective speed of 300 MHz. As before, each tested algorithm was found to perform better than the others for some subset of the considered scenarios. Particularly, it was observed that global algorithms are heavily affected by run-queue-related overheads. C-EDF exhibited schedulability in the HRT case that is intermediate between G-EDF and P-EDF. In the SRT case, C-EDF generally exhibited the best schedulability, as well as lower tardiness than G-EDF.

In a third study [9], Brandenburg and Anderson evaluated seven possible implementations of G-EDF in LITMUS$^{\mathrm{RT}}$ on the above-mentioned Niagara platform. Tradeoffs involving implementation approaches were found to significantly impact schedulability.

The idea of a clustered approach to ameliorate limitations of partitioned and global approaches on large multiprocessor platforms was introduced by Calandrino *et al.* [13] and Baker and Baruah [3]. Notably, Calandrino *et al.* presented guidelines for defining clusters for SRT workloads. Empirical results were obtained by them using the SESC architecture simulator for a 64-core platform.

Shin *et al.* [25] presented a study concerning virtual clusters. Virtual clusters can share processors of the underlying platform, while physical clusters are completely independent. In this paper, we focus on physical clusters only.

Contributions. Questions regarding the implementation of clustered algorithms and the performance of such algorithms in comparison to global and partitioned algorithms have not been fully answered by previous studies. In [13], clustering is considered only in the context of SRT systems and the presented evaluation is based on an architecture simulator. In [11], due to architectural limitations, only one cluster size could be considered. If multiple levels of shared cache exist, it is not clear which level should be used for clustering. Additionally, preemption/migration costs are assumed in [11] based on a *fixed* per-job working set size. It is not clear whether similar conclusions would have been reached for other cost choices.

In this paper, we present an empirical comparison of G-EDF, P-EDF, and C-EDF where clustering issues, runtime overheads, and cache-related costs are explicitly considered. Overheads were measured on a large (by today's standards) 24-core Intel Xeon platform with a three-level cache hierarchy that enables the evaluation of *two* different cluster sizes (at the L2 and L3 cache levels). Scheduling algorithms are compared on the basis of real-time *schedulability*, assuming measured runtime overheads and cache-access costs. Furthermore, in contrast to previous studies, proposed real-time schedulability tests are compared to "brute-force" tests to assess their pessimism. Finally, this is the first in-depth study to use a new approach for addressing preemption/migration costs that allows a wide range of tradeoffs involving such costs to be considered.

Our major findings are as follows: **(i)** in the HRT case, G-EDF was never preferable on our platform, and P-EDF outperformed all other tested algorithms even assuming unrealistically high preemption costs; **(ii)** proposed HRT schedulability analysis for global and clustered approaches was significantly more pessimistic than corresponding "brute-force" tests, which in turn were inferior to P-EDF analysis; **(iii)** in the HRT case, "less global" approaches (P-EDF and C-EDF-L2, which clusters at the L2 level) were consistently better than "more global" approaches (G-EDF and C-EDF-L3, which clusters at the L3 level); **(iv)** bin-packing issues were mostly negligible for clusters of size six; **(v)** C-EDF (both L2 and L3 variants) proved to be particularly effective in the SRT case. Given these results, we believe that future global scheduling research should focus on devising better schedulability analysis for small-to-medium multicore platforms rather than large multicore platforms.

In the sections that follow, we provide needed background (Sec. 2), discuss methodology and architectural considerations related to overhead measurement (Sec. 3), present our schedulability experiments and our findings (Sec. 4), and conclude (Sec. 5).

2 Background

We focus herein on the scheduling of a system of *sporadic tasks*, $T_1, \ldots, T_n$, on m identical processors $P_1, \ldots, P_m$. Each task T_i is specified by its *worst-case execution time* e_i and its *period* p_i. The j^{th} job of task T_i is denoted T_i^j. Such a job T_i^j becomes available for execution at its *release time* r_i^j and should complete by its *deadline* $r_i^j + p_i$; otherwise it is *tardy*. The spacing between r_i^j and r_i^{j+1} must satisfy $r_i^{j+1} \geq r_i^j + p_i$. A tardy job T_i^j does not alter r_i^{j+1}, but T_i^{j+1} cannot execute until T_i^j completes. Task T_i's *utilization*, e_i/p_i, reflects the processor share it requires; the sum $\sum_{i=1}^{n} e_i/p_i$ denotes the *total utilization* of the system.

Scheduling. A HRT system is considered to be *schedulable* iff it can be shown that no job deadline is ever missed. A SRT system is considered (in this paper) to be *schedulable* iff it can be shown that deadline tardiness is bounded. Methodologies for checking schedulability necessarily depend on the evaluated scheduling algorithm and overheads that arise in practice, such as context-switching times, cache-related delays, *etc.* Such overheads are typically accounted for by inflating per-job execution costs.

We investigate G-EDF, P-EDF, and C-EDF as representatives of different multiprocessor approaches in the class of preemptive, priority-driven, work-conserving real-time scheduling algorithms.[1] In EDF scheduling algorithms, jobs are scheduled in order of increasing deadlines, with ties broken arbitrarily. In C-EDF, tasks are statically assigned to fixed-sized clusters and globally scheduled within each cluster. P-EDF and G-EDF can be seen as special cases of C-EDF: in P-EDF, each cluster consists of only one core, while in G-EDF, all cores form one cluster. If $m > 1$, deadline misses can occur under each EDF variant in feasible systems (*i.e.*, systems with total utilization at most the number of processors). Therefore, restrictive caps on total utilization are required under each considered scheduling approach to ensure timing constraints in the HRT case. In contrast, G-EDF ensures bounded tardiness in the SRT case as long as the system is not overutilized [15]. C-EDF was proposed as a compromise between these two approaches for large multicore platforms where a hierarchy of cache memories is employed. On such platforms, caches are organized in levels where the fastest (and usually smallest) caches are denoted as *level-1* (L1) caches, with deeper caches (L2, L3, *etc.*) being successively larger and slower. Generally, L1 caches are private per-core caches, while L2 and L3 caches are shared among a progressively larger number of cores. In C-EDF, all cores that share a specific cache level (L2 or L3) are defined to be a cluster; tasks are allowed to migrate within a cluster, but not across clusters. Clustering lowers migration costs and lessens run-queue contention in comparison to G-EDF, and eases bin-packing problems associated with P-EDF. In particular, bin packing becomes easier because clustering results in fewer, larger bins. Under C-EDF, deadline tardiness is bounded if the total utilization of the tasks assigned to each cluster is at most the number of cores per cluster. We use the notation C-EDF-L2 (C-EDF-L3) when we wish to specifically indicate that each cluster is defined to include all cores that share an L2 (L3) cache.

LITMUS$^{\mathrm{RT}}$. As mentioned in Sec. 1, LITMUS$^{\mathrm{RT}}$ is a real-time extension of the Linux kernel. The version of LITMUS$^{\mathrm{RT}}$ used in this study was obtained by rebasing to the Linux 2.6.32 kernel (from 2.6.24) and porting to the x86_64 architecture. In LITMUS$^{\mathrm{RT}}$, scheduling policies are implemented as *plugins* that can be activated at runtime (see [11] for an in-depth description). LITMUS$^{\mathrm{RT}}$ supports both event- and quantum-driven scheduling, but in this study, we only consider event-driven scheduling.

[1] "Work-conserving" is interpreted with regard to a cluster (resp., partition) under C-EDF (resp., P-EDF).

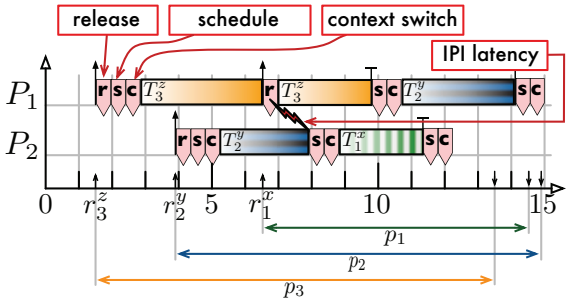

Figure 1: Example G-EDF schedule with overheads for three jobs (T_1^x, T_2^y, T_3^z), where $(e_i, p_i)_i = (2.5, 8)_1, (6, 11)_2$, and $(6.5, 12)_3$, on two processors (P_1, P_2). Large up-arrows denote interrupts, small up-arrows denote job releases, down-arrows denote job deadlines, T-shaped arrows denote job completions, and wedged boxes denote overheads (which are magnified for clarity). Job releases occur at $r_3^z = 1.5, r_2^y = 3.9$, and $r_1^x = 6.5$.

From the previous version of LITMUS$^{\mathrm{RT}}$, the C-EDF plugin has undergone several modifications to better support runtime clustering around specific cache levels. In the current version, C-EDF can automatically detect the cache hierarchy of the majority of recent CPU models and can automatically identify which cores share a specific cache level. Identifying this topology is non-trivial in Linux as the assignment of cpu-id numbers does not necessarily reflect the system's cache layout. The current C-EDF implementation supports runtime cluster-size changes (when no real-time workload is present).

In the comparison of G-EDF implementations given in [9], the best performing implementation — which makes use of coarse-grained locking to synchronize ready-queue accesses — used dedicated interrupt handling (*i.e.*, a single core is exclusively reserved for interrupt processing). In the present study, dedicated interrupt handling was not used. This choice was made to allow a better comparison between G-EDF and C-EDF. Dedicated interrupt handling prevents the scheduling of real-time jobs on the CPU that manages interrupts, and this imposes unacceptable limitations on small clusters.

Overheads. In LITMUS$^{\mathrm{RT}}$, tasks are delayed by six major sources of overhead, four of which are illustrated in Fig. 1. When a job is released, *release overhead* is incurred, which is the time needed to service the interrupt routine that is responsible for releasing jobs at the correct times. Whenever a scheduling decision is made, *scheduling overhead* is incurred while selecting the next process to execute and requeuing the previously-scheduled process. Switching the execution stack and processor registers causes a *context-switch overhead*. These overhead sources occur in sequence

in Fig. 1, on processor P_1 at time 1.5 when T_3^z is released, and again on processor P_2 at time 3.9 when T_2^y is released. *Inter-processor interrupt (IPI) latency* is a source of overhead that occurs when a job is released on a processor that differs from the one that will schedule it. This situation is depicted in Fig. 1, where at time 6.5, T_1^x is released on P_1, which triggers a preemption on P_2 by sending an IPI. *Tick overhead* (the time needed to manage the interrupt timer) occurs to a very limited extent under event-driven scheduling and is therefore not shown in Fig. 1. On real systems, *preemption* and *migration overhead* (or delay) account for any costs due to a loss of cache affinity. Preemption (resp., migration) overhead is incurred when a preempted job later resumes execution on the same (resp., different) processor. These overheads are analyzed in more depth below.

3 Overhead Measurements

In Sec. 4, we report on schedulability experiments that were conducted with real overheads considered. In this section, we describe how such overheads were determined.

Runtime overheads and cache-related preemption and migration delays can be accounted for in scheduling analysis by inflating per-task execution costs. For the algorithms of interest in this paper, overheads that occur before or after a job is scheduled can be accounted for by extending the job's execution: a job incurs two scheduling and context-switching overheads [21] and one preemption/migration overhead.[2] Similar accounting can be done for IPI latencies and a job's own release overhead. An in-depth discussion of these techniques is presented in [15]. Accounting for release overheads due to *other* jobs is more problematic as their occurrence is interrupt-based. A detailed discussion of this topic can be found in [12].

We measured runtime overheads and preemption/migration delays in LITMUS$^{\mathrm{RT}}$ on an Intel Xeon L7455 system. The L7455 is a 24-core 64-bit uniform memory access (UMA) machine with four physical sockets. Each socket contains six cores running at 2.13 GHz. All cores in a socket share a unified 12-way set associative 12 MB L3 cache, while groups of two cores share a unified 12-way set associative 3 MB L2 cache. Each core also includes an 8-way set associative 32 KB L1 data cache and an identical L1 instruction cache. The three-level cache hierarchy of our machine allowed us to set two cluster sizes for C-EDF. In the first configuration, C-EDF-L2, cores are grouped around L2 caches and the platform is partitioned into 12 clusters of two cores each. The second configuration, C-EDF-L3, groups cores around L3 caches, partitioning the platform into four six-core clusters.

[2] Any lack of cache-affinity in a newly-released job J_i is already accounted for in e_i. Therefore, only the preempted job (if any) must be considered.

3.1 Kernel Overheads

Measuring kernel overheads is not as straightforward as it may seem. The Linux kernel contains several sources of unpredictability (such as interrupt handling and priority inversion) and our hardware platform (as the vast majority of platforms on which Linux runs) lacks the determinism expected in HRT environments. Nonetheless, it has been claimed that Linux can handle a large and important subset of real-time applications [22] and LITMUS$^{\mathrm{RT}}$ objectives are in accordance with this claim. Despite advances made in recent years to bound migration delays and analyze interferences due to shared hardware resources [24, 27], it is currently very difficult to determine *verifiable* worst-case overhead bounds. In fact, on multicore platform with a complex hierarchy of shared caches, current timing analysis tools are not yet able to analyze complex interactions between tasks that arise due to atomic operations, bus locking, and bus and cache contention [26]. Thus, kernel overheads must be determined experimentally, through a repeated measurement process. We used Feather-Trace [10], a light-weight cycle-counter-based tracing tool to obtain such overheads, following the same methodology previously described in [9, 11]. A small number of samples collected in the measurement process may be "outliers," due to the lack of determinism noted above. To account for this, before computing maxima and averages, we applied a 1.5 interquartile range (IQR) outliers removal technique.[3]

For each algorithm, runtime overheads were obtained by measuring the system's behavior for periodic task sets. As G-EDF and P-EDF are a special case of C-EDF, task sets were defined *per-cluster*, using a single cluster of size 24 for G-EDF, 24 clusters of size one for P-EDF, and so on. Per-cluster task-set sizes were defined to range over [10, 350] for G-EDF (task-set sizes are equal to the total number of tasks in this case), over [1, 15] for P-EDF, over [3, 50] for C-EDF-L3, and over [1, 20] for C-EDF-L2. The granularity of each range is defined by steps that were sized variably to achieve higher resolution when the total number of tasks is at most 60 (the majority for task sets for the distributions presented in Sec. 4 have sizes in the range [1,60]).

For each task-set size, we measured 10 task sets generated randomly (with uniform light utilizations and moderate periods[4]; see Sec. 4), for a total of 550 task sets. Each task set was traced for 60 seconds. In total, more than 35 GB of trace data and 500 million individual overhead measurements where obtained during more than 20 hours of tracing. After removing outliers as discussed above, we computed average- and worst-case overheads for each plugin as a function of task-set size, which resulted in 12 graphs.

[3] According to this technique, an outlier is a sample that falls more than 1.5 IQR below the first quartile or above the third quartile. NIST [23] suggests the use of IQR as a standard technique for removing outliers.

[4] We further verified that overheads measured using task sets generated with uniform medium utilizations yielded similar results.

Due to space constraints, we only discuss here the two representative graphs shown in Fig. 2 (a complete set of graphs for all measured kernel overheads can be found in [5]). Inset (a) of the figure shows worst-case scheduling overheads (measured in μs) as a function of the total number of tasks. The most notable trend here is the high scheduling overhead (up to 200 μs) incurred by G-EDF compared to the overhead experienced by C-EDF and P-EDF (less than 30 μs). Scheduling overhead for G-EDF sharply increases with the number of tasks, as the contention and the length of the global run queue increases. Such overhead is likely due to the cost of frequent cache line migrations ("cache-line bouncing") and heavy contention for the global run-queue lock among all cores. Interestingly, scheduling overhead for the C-EDF variants and P-EDF are similar.

Fig. 2 (b) shows worst-case release overheads as a function of the total number of tasks. As before, G-EDF overhead is remarkably higher than those of the other plugins. Again, this is mostly due to cache-line-bouncing effects and to the higher contention for the global run queue. Interestingly, P-EDF overhead is markedly lower than C-EDF overhead and C-EDF-L3 overhead is more dependent on the number of tasks.

We used monotonic piecewise linear interpolation to derive upper bounds for each plugin and for each overhead as a function of the task set size. These upper bounds were used in the schedulability experiments described in Sec. 4. For the few overheads where the measurements did not reveal a conclusive trend, we assumed a constant value equal to the maximum observed value.

3.2 Cache-Related Delays

When a job resumes execution after a preemption or a migration, it is likely to suffer additional cache misses (and thus extra delays) due to the perturbation of caches while the job was not scheduled [20]. In such a situation, we say that the job has lost *cache affinity*. Cache-related delays clearly depend on task working set sizes (WSSs) and possibly on the scheduling algorithm and on the task set size (TSS), but their measurement is a difficult problem [11].

In a recent paper [6], we conducted a detailed investigation of cache-related preemption and migration delays (CPMD) and proposed two methods for their empirical measurement. The *schedule-sensitive* method aims to detect dependencies of CPMD on the scheduling policy and on TSS. In this method, delays are recorded on-line by executing (under the desired scheduling policy) test cases with a wide range of TSSs and WSSs. The main drawback of this method is that it is not possible to control *when* a preemption or a migration will happen, and therefore many of the recorded delays are discarded as invalid (*e.g.*, a job may not be preempted at all, or may be preempted prematurely, thus preventing a complete and valid measurement to be taken). The *synthetic* method overcomes this problem by

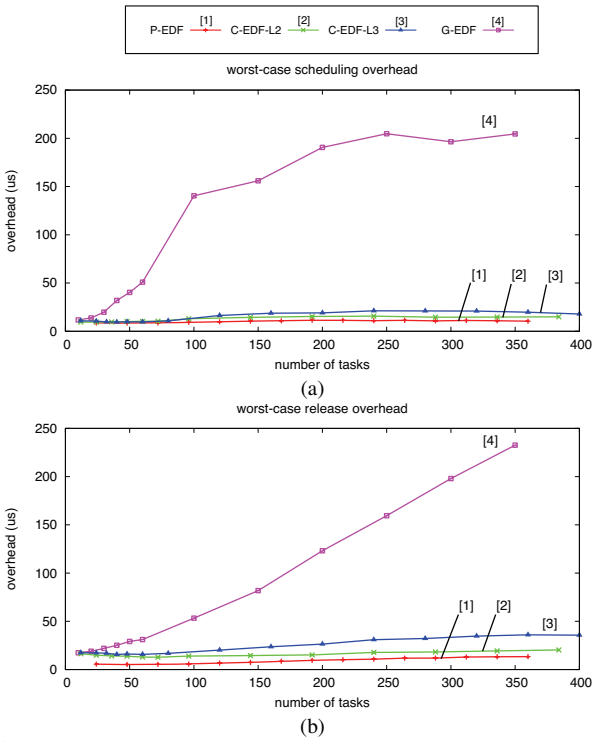

Figure 2: Sample worst-case overhead measurements. The graphs show worst-case measured overheads (in μs) as function of task set size. **(a)** Scheduling overhead. **(b)** Release overhead.

achieving finer control over the measurement process by artificially triggering preemptions and migrations of a single task. Under this method, a fixed-priority scheduling policy (such as POSIX SCHED_FIFO) is employed and a single highest-priority task repeatedly accesses working sets of different sizes. In the synthetic method, preemptions and various types of migrations are explicitly controlled and a high number of CPMD measurements of each kind are collected. As this method does not depend on the scheduling policy employed, it cannot detect dependencies on TSS.

In [6], we measured CPMDs using both methods for many combinations of TSSs and WSSs, for different read/write ratios, and in two system configurations (with or without a background workload consisting of low-priority cache-polluter jobs). The platform used in that study is *the same* Intel Xeon platform used in this paper. Our experiments showed that CPMD depends on preemption length, but *does not depend on task set size*. Our findings also showed that on our platform, CPMD in a system under load is only predictable for WSSs that *do not thrash the L2 cache*. Furthermore, preemption and migration delays *do not differ significantly* for a system under load. These two trends can be clearly seen in Fig. 3, which reports CPMDs obtained with the synthetic method in a system with a background cache-polluting workload. The figure shows average CPMD as a function of WSS, assuming a read/write ratio of 75/25. CPMD values are shown (in a bar chart) for preemptions and the different kinds of migrations that can

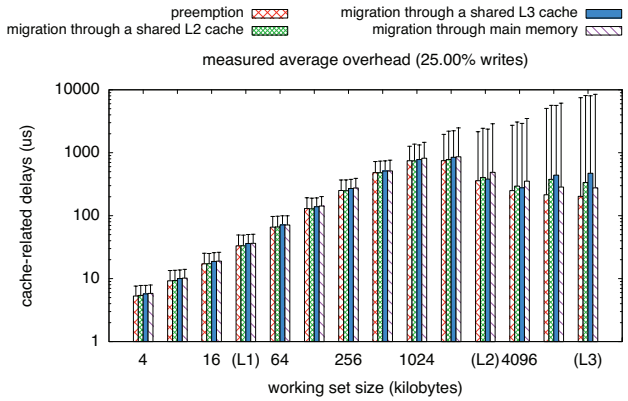

Figure 3: CPMD delays obtained with the synthetic method in a system with high cache contention. The graphs show average CPMD (in μs) for preemptions and different types of migrations as a function of WSS (in KB). The error bars indicate one standard deviation.

take place on our platform: migrations through L2 cache, through L3 cache, and through main memory. As can be seen in Fig. 3, when the WSS approaches the size of the L2 cache (3072 KB), the standard deviation, depicted as error bars in the figure, becomes very large (note that the scale is logarithmic) and therefore overhead measurements are very imprecise. Furthermore, Fig. 3 shows that there is no substantial difference between preemption and migration costs in a system under load.

4 Schedulability Experiments

We compared G-EDF, P-EDF, C-EDF-L2, and C-EDF-L3 on the basis of schedulability, which was assessed by generating task sets at random using distributions similar to those proposed by Baker [2]. An algorithm's *schedulability* (HRT or SRT) is defined as the fraction of generated task sets that are schedulable (HRT or SRT) under it. Similarly to [9, 11, 14], we used three period and six utilization distributions. Task utilizations were generated using three uniform and three bimodal distributions. The ranges for the uniform distributions were [0.001, 0.1] (*light*), [0.1, 0.4] (*medium*), and [0.5, 0.9] (*heavy*). For the three bimodal distributions, utilizations uniformly ranged over either [0.001, 0.5) or [0.5, 0.9] with respective probabilities of 8/9 and 1/9 (*light*), 6/9 and 3/9 (*medium*), and 4/9 and 5/9 (*heavy*). Task periods were generated similarly, using three uniform distributions with ranges [$3ms$, $33ms$] (*short*), [$10ms$, $100ms$] (*moderate*), and [$50ms$, $250ms$] (*long*). All periods were chosen to be integral.

Tasks were created by choosing utilizations and periods from their respective distributions and computing execution costs. Each task set was generated by creating tasks until the total utilization exceeded a specified cap (varied between 1 and 24) and by then discarding the last-added task, to allow for some slack for overheads.

4.1 Performance Metric

Prior to testing schedulability, task parameters were inflated to account for the overheads described in Sec. 3. Maximum overheads were used in the HRT case, while average overheads were used in the SRT case. As noted in [9, 11, 14], kernel overheads should be accounted for after a task set has been generated, as such overheads are mostly TSS-dependent. In contrast, CPMDs are independent of TSS, but correctly devising a bound for these overheads requires knowledge of task's WSS [6]. Anticipating realistic WSS distributions is a non-trivial problem, and therefore, previous studies [9, 11, 14] focused on *selected* WSSs. In contrast, instead of assuming specific WSSs and deriving corresponding cache-related delays, in this study we account for such delays by adopting a *WSS-agnostic approach* [6]. In this approach, CPMD is an additional parameter of the task generation procedure and schedulability becomes a function of two variables: the cap U on total utilization and the CPMD D. Schedulability can therefore be studied assuming a broad range of values for D (and thus WSS).

While the WSS-agnostic method is appealing for its avoidance of a specific WSS bias, presenting schedulability results for this approach poses some practical problems. In particular, displaying results for the two-variable (U and D) schedulability function requires 3D graphs. As such plots need to display at least four curves (one for each tested algorithm), the resulting graphs may be very difficult to interpret. To overcome this limitation, we proposed a new aggregate performance metric [6].

Weighted schedulability [6]. Let $S(U, D) \in [0, 1]$ denote an algorithm's schedulability for a given U and D, and let Q denote a set of evenly-spaced utilization caps (*e.g.*, in our setup, $Q = \{1.0, 1.25, 1.50, \ldots, 24\}$). Then *weighted schedulability* $W(D)$ is defined as

$$W(D) = \frac{\sum_{U \in Q} U \cdot S(U, D)}{\sum_{U \in Q} U} \quad .$$

Weighting individual schedulability results by U reflects the intuition that high-utilization task systems have higher "value" since they are more difficult to schedule. This metric reduces schedulability results to a 2D (and thus easier to interpret) plot without fixing a particular WSS or a specific utilization cap. Furthermore, it exposes *ranges* of CPMD where a particular scheduler is competitive. $W(D)$ can reveal interesting tradeoffs that cannot be easily inferred from fixed-CPMD schedulability. This is illustrated in the following example.

Fig. 4 shows a comparison between fixed-CPMD schedulability results and $W(D)$ results. Both insets refer to SRT schedulability and were obtained for the short-period/heavy-utilization case.[5] Fig. 4(a) indicates the fraction of generated task sets each algorithm successfully

[5]This case is interesting since video playback and interactive games of-

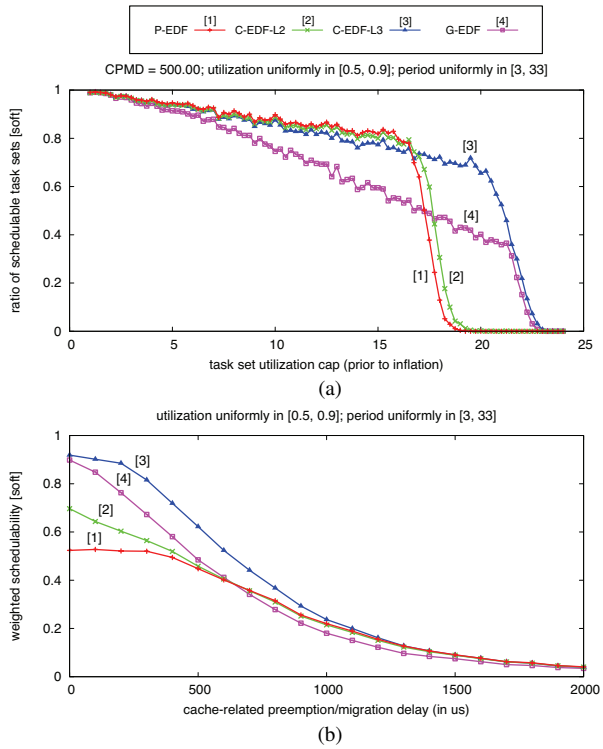

Figure 4: Comparison of fixed-CPMD and $W(D)$ schedulability. The graphs show SRT schedulability for the uniform heavy utilization distribution with short periods. **(a)** Schedulability as function of U for CPMD = $500\mu s$. **(b)** $W(D)$ schedulability as function of CPMD. Note the different X-axis in each inset.

scheduled, as a function of total utilization and assuming a fixed D of $500\mu s$. As can be observed in Fig. 3, this cache-related delay is the average delay experienced by tasks with a WSS of 512 KB on our platform for both preemptions and migrations when the system is under load. As can be seen, C-EDF-L3 is the best performing algorithm. C-EDF-L2 and P-EDF exhibit similar, but worse, performance. These trends arise if $D = 500\mu s$ for both preemption and migration costs, but if preemptions were considerably cheaper than migrations, would P-EDF perform better than C-EDF-L3? Fig. 4 (a) does not give any insight as to how to answer this question.

In contrast, the $W(D)$ graph in Fig. 4 (b) provides an immediate answer. Inset (a) collapses to four distinct points (one for each algorithm) at $D = 500$ in inset (b). Inset (b) indicates weighted schedulability as a function of CPMD D. In this plot, the curve for C-EDF-L3 reveals that for $D \leq 600\mu s$, C-EDF-L3 is superior to P-EDF. Therefore, if the cost of migrations is less than $600\mu s$, C-EDF-L3 should be preferred to P-EDF, even if the cost of preemptions is $0\mu s$ (the interested reader is referred to [6] for a more in-depth survey of weighted schedulability). We believe that weighted schedulability plots are a valuable aid that may help practitioners to select an appropriate real-time scheduler to use, basing the choice on actual measured CPMD values.

ten fall into the period range $[3ms, 33ms]$, and high-definition multimedia processing is likely to cause heavy utilizations.

4.2 Schedulability Tests

In this study, we chose to vary D over $[0\mu s, 2000\mu s]$. This range of values seems reasonable on a platform like ours, where (as noted in Sec. 3) cache-related delays are non-predictable for WSSs that exceed the size of L2 cache and the average-case delay for a 1024 KB WSS in a system under load is approximately $1000\mu s$.

Schedulability was checked for different categories of task systems under P-EDF, G-EDF, C-EDF-L2, and C-EDF-L3. For P-EDF and both variants of C-EDF, we determined whether each task set could be partitioned using the *worst-fit decreasing heuristic*. Under P-EDF, HRT and SRT schedulability differ only in the use of maximum or average overheads: under partitioning, if tardiness is bounded, then it is zero, so the only way to schedule a SRT task set is to view it as hard. Under C-EDF, schedulability for each cluster was checked by applying the appropriate G-EDF test (HRT or SRT) within the cluster.

HRT schedulability under G-EDF (C-EDF) was determined by testing whether a given task set (cluster) passes at least one of five major sufficient — but not necessary — HRT schedulability test [1, 4, 7, 8, 17]. For SRT schedulability, since G-EDF can guarantee bounded deadline tardiness if the system is not overloaded [15], only a check that total utilization is at most m (the number of processors) is required.

In this study, we further compared HRT schedulability results with results obtained from "brute-force" (BF) schedulability tests. In such a test, simulation (with overheads considered) is used to produce a periodic schedule, and a task set is deemed to be unschedulable if any deadline misses are found. Given the very large hyperperiods of the generated task sets, exhaustive simulations were infeasible. Thus, each task set was simulated for 60 seconds or until a deadline miss was found. Results from such BF tests represent an upper bound on schedulability for each tested algorithm: task sets claimed unschedulable by a BF test are certainly not schedulable, while a BF test may wrongly claim as schedulable task sets that miss a deadline at a later point in the schedule, or that exhibit deadline misses only under non-periodic arrival sequences. Note that, since EDF is optimal on uniprocessors, a BF test is not of interest for P-EDF: if a task set can be successfully partitioned onto individual processors, then no job will miss a deadline, and simulating a schedule is therefore pointless. For each algorithm, and for each (U, D) pair, we compared $W(D)$ and BF (except for P-EDF) by testing 1,000 task sets. Considering a spacing of 0.25 for U and a spacing of $100\mu s$ for D, more than 7.5 million task sets were evaluated.

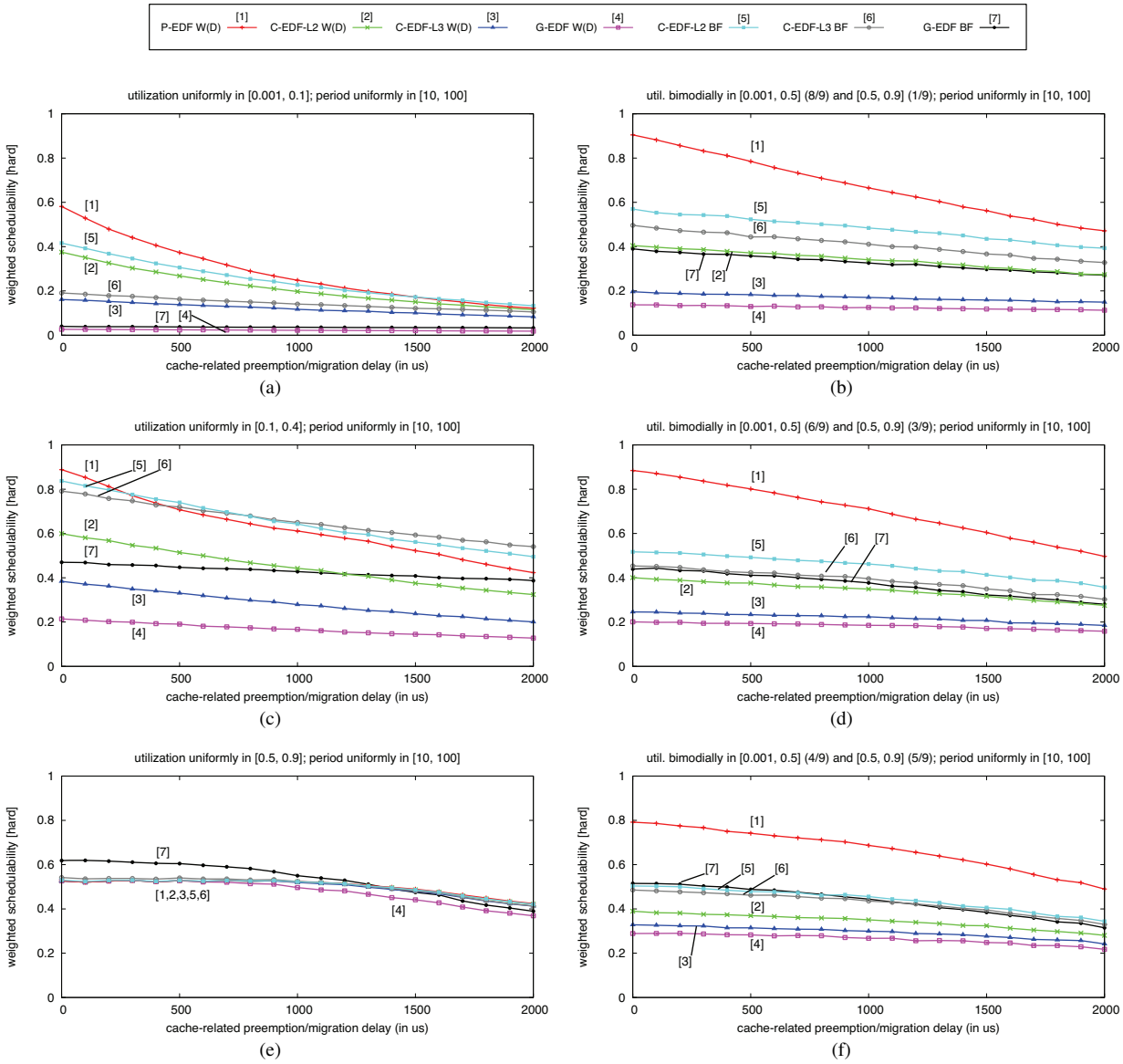

Figure 5: HRT weighted schedulability for moderate periods as a function of cache-related preemption/migration cost for various utilization distributions. **(a)** Uniform light. **(c)** Uniform medium. **(e)** Uniform heavy. **(b)** Bimodal light. **(d)** Bimodal medium. **(f)** Bimodal heavy. Recall that CPMD is the extra delay a job incurs due to a loss of cache affinity when resuming execution after a preemption or a migration. Note that these graphs allow meaningful comparisons between different x-coordinates — *e.g.*, a particular workload may incur only $200\mu s$ CPMD under C-EDF and $400\mu s$ under G-EDF (see Sec. 4.1 for an in-depth explanation of weighted schedulability).

4.3 Results

HRT $W(D)$ results for the moderate period distributions are shown in Fig. 5. The first column of the figure (insets (a,c,e)) gives results for the three uniform distributions (light, medium, heavy) and the second column (insets (b,d,f)) gives results for the three bimodal distributions. The plots indicate both $W(D)$ and BF test results for D ranging over [0, $2000\mu s$]. Weighted schedulability results for the SRT case are shown in Fig. 6, which is organized similarly to Fig. 5. Due to space constraints, other graphs (over 700 of them) are not shown here but can be found in the extended version of the paper [5].

Observation 1. G-EDF *is never preferable for HRT on our platform.* In fact, Fig. 5 shows that the $W(D)$ curve for P-EDF dominates the BF curves of all the other algorithms in most graphs, independently of preemption/migration costs. The BF test is *optimistic* and represents an upper bound on the schedulability of G-EDF and C-EDF. Therefore, even if a perfect G-EDF feasibility test (*i.e.*, a test that never wrongly claims a schedulable task set as unschedulable) were employed, G-EDF would still not be preferable to P-EDF in most cases, even when assuming unrealistically low

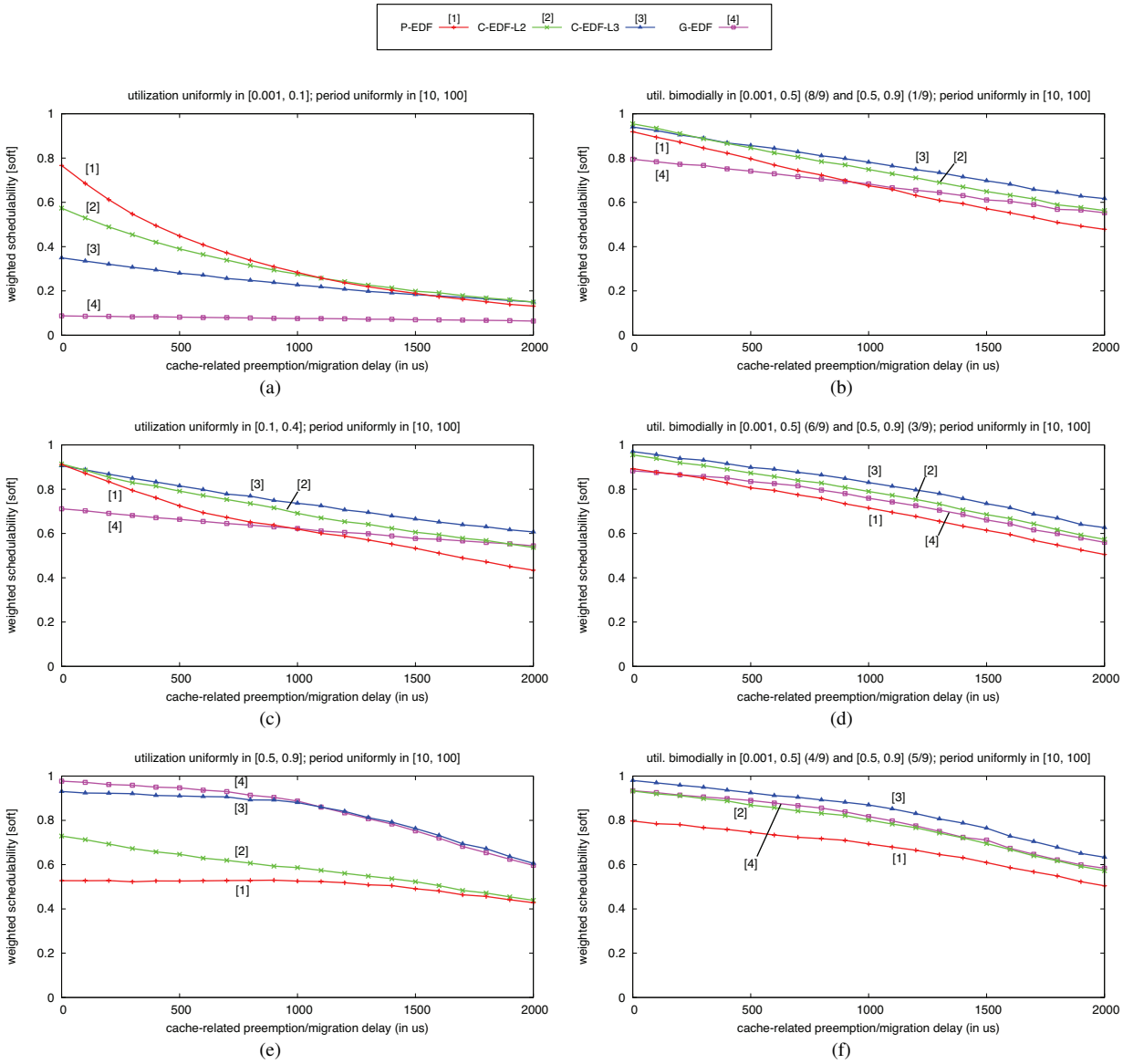

Figure 6: SRT weighted schedulability for moderate periods as a function of cache-related preemption/migration cost for various utilization distributions. **(a)** Uniform light. **(c)** Uniform medium. **(e)** Uniform heavy. **(b)** Bimodal light. **(d)** Bimodal medium. **(f)** Bimodal heavy. Recall that CPMD is the extra delay a job incurs due to a loss of cache affinity when resuming execution after a preemption or a migration. As noted in Fig. 5, these graphs allow meaningful comparisons between different x-coordinates.

CPMD for G-EDF. In other words, *existing* P-EDF analysis is superior to any *yet-to-be developed* G-EDF analysis in most of the considered scenarios. This strongly calls into question the viability of G-EDF as a HRT scheduler.

Observation 2. *Pessimistic HRT schedulability analysis strongly impacts C-EDF.* Fig. 5 shows that both C-EDF-L2 and C-EDF-L3 are never preferable to P-EDF when using existing analysis ($W(D)$ curves). Furthermore, even when assuming perfect analysis, C-EDF-L2 and C-EDF-L3 remain inferior to P-EDF in most cases. In fact, the C-EDF upper bound indicated by the BF curves is inferior to P-EDF in all but inset (c) of Fig. 5.

Observation 3. *Among non-partitioned approaches, C-EDF-L2 is superior in the HRT case.* The $W(D)$ and BF test results (Fig. 5, all insets) indicate that C-EDF-L2 performs consistently better than C-EDF-L3 and G-EDF (or, at most, comparably, inset (e)). This confirms the idea that "more global" EDF schedulers are inferior in the HRT case.

This observation, together with Obs. 1 and Obs. 2 above, suggests that future improvements in G-EDF analysis (which is used to evaluate intra-cluster C-EDF schedulability) should focus on *enhancing schedulability bounds for small-to-medium clusters*. Nonetheless, our results indicate that even major improvements in G-EDF analysis

could only lead to modest gains in HRT schedulability. In fact, as noted in Obs. 2, even perfect G-EDF analysis would make C-EDF-L2 preferable to P-EDF only in a small subset of the considered scenarios (Fig. 5 (c)).

Observation 4. *Conservative HRT schedulability analysis heavily underestimates the performance of global schedulers in high-variance utilization distribution scenarios.* In the HRT case, the gap between $W(D)$ and BF is small for uniform light and heavy utilizations (Fig. 5(a,e)). In contrast, in the uniform medium and in all bimodal cases, the $W(D)$ results are significantly inferior to the BF results (Fig. 5, insets (b,c,d,f)). In such cases (which are perhaps more representative of real-world workloads), task utilizations may vary greatly. Existing G-EDF analysis is unable to fully characterize such variations and therefore the $W(D)$ results for G-EDF and C-EDF are noticeably lower than the BF results. Instead, when utilization is uniformly light, all task sets are comprised of many small tasks, which are significantly affected by overheads. In this context, all algorithms perform poorly, and conservative schedulability bounds are quite close (Fig. 5(a)). In the uniform heavy utilization case, overheads have a minor impact on the few large tasks that compose each task set. Nonetheless, given their utilizations, such task sets are difficult to schedule and $W(D)$ correctly approximates BF (Fig. 5(e)).

Observation 5. *Bin-packing limitations are mostly negligible for clusters of size six.* If the system is not overutilized, G-EDF is optimal for SRT (*i.e.*, G-EDF guarantees bounded tardiness for any task set with total utilization at most m); therefore, if a task set can be partitioned under C-EDF, then it is schedulable for SRT. Since the schedulability test for SRT is not pessimistic, Fig. 6 (which reports SRT $W(D)$ results) reveals a tradeoff between bin-packing and overheads. When bin-packing limitations are not an issue (because all task utilizations are small — Fig. 6(a)), lower overheads favor P-EDF and C-EDF-L2 over C-EDF-L3 and G-EDF. Instead, bin-packing limitations clearly affect P-EDF when task utilizations are heavy: Fig. 6(e,f) shows that a small increase in cluster size (from one core — P-EDF— to two cores — C-EDF-L2) is sufficient to boost performance. In particular, Fig. 6 shows that bin-packing is not a limitation for C-EDF-L3: $W(D)$ curves for C-EDF-L3 are as high as G-EDF curves in all insets. Since bin-packing problems become easier with larger and fewer bins, *clusters of size six (or larger) are sufficient to avoid bin-packing limitations in the majority of the tested scenarios*. Previous studies based on an architecture simulator [13] have shown that a cluster size of four may be sufficient to avoid bin-packing issues, but, due to the topology of our platform, such cluster size is not desirable. Given such results, we believe that future work on improving SRT tardiness bounds should focus on platforms with at most four to eight cores.

Observation 6. *The C-EDF approaches are superior to the other algorithms in the SRT case.* In the majority of the tested scenarios (Fig. 6), C-EDF-L3 and C-EDF-L2 usually performed better than G-EDF and P-EDF even under moderate-to-high migration costs and low preemption costs. For example, in Fig. 6(b), the C-EDF-L3 and C-EDF-L2 curves are as high as the P-EDF curve even assuming $200\mu s$ for migrations and $0\mu s$ for preemptions! C-EDF-L3 generally exhibits slightly higher schedulability than C-EDF-L2, while both C-EDF approaches have lower overheads than G-EDF.

Most of the points made above have not been noted in prior studies; those that have strengthen earlier findings [11] by eliminating a specific WSS bias.

5 Conclusion

In this paper, we presented an empirical comparison of G-EDF, P-EDF, and C-EDF (with two cluster sizes) multiprocessor real-time scheduling algorithms on a large 24-core Intel platform. Scheduling algorithms were compared on the basis of real-time *schedulability*, assuming measured runtime overheads and cache-related delays. In contrast with previous studies, schedulability test results were compared to results from simulation-based "brute force" tests to assess the pessimism of the currently-available tests. A new aggregate performance metric (*weighted schedulability*) was used to easily compare schedulability results for a wide range of cache-related delays and to clearly expose the range of CPMDs in which a particular scheduler is competitive. To the best of our knowledge, this study is the first to present a comparison of C-EDF real-time schedulability for multiple cluster sizes assuming real hardware overheads.

Our results indicate that G-EDF is never preferable for HRT. In most of the considered scenarios, *existing P-EDF analysis was superior not only to existing G-EDF analysis, but also to optimistic upper bounds for any yet-to-be-developed G-EDF analysis*. Furthermore, if the cost of migrations is non-negligible, then the high schedulability gap between P-EDF and the other tested algorithms calls into question the benefits of migrations for EDF scheduling algorithms in the HRT case on large multicore platforms. In the HRT case, C-EDF-L2 performed best among the non-partitioned algorithms, although the pessimism of G-EDF analysis (particularly for workloads with high-variance utilization distribution) strongly impacted C-EDF. Our results also show that, practically speaking, *bin-packing limitations are negligible for clusters of size six*. Therefore, our findings suggest that *future HRT global scheduling research should focus on small/medium-sized platforms*. In the SRT case, the C-EDF approaches (C-EDF-L3 in particular) were superior to all other evaluated algorithms. Thus, future work on tardiness bounds under G-EDF should focus on low processor counts (four to eight), as they are encoun-

tered under C-EDF during intra-cluster analysis.

Limitations. The topology and the cache layout of the platform used in our experiments prevented us from testing cluster sizes different from two (C-EDF-L2) and six (C-EDF-L3). It would be interesting to extend the experiments proposed in this paper using platforms with a different cache layout (*e.g.*, on platforms that enable cluster sizes of four/eight) or using an architecture simulator (*e.g.*, to test the impact of an additional cache level).

Our G-EDF and C-EDF implementations make use of locks to synchronize accesses to the ready queue. This inevitably causes overheads due to contention for the ready queue. Although it would be interesting to repeat the study described in this paper using a *non-blocking* ready-queue implementation, to the best of our knowledge, all published algorithms that support non-blocking priority queues require either bounded priority ranges (G-EDF requires unbounded priorities) or multi-word compare-and-swap instructions (not supported by our hardware). Furthermore, for a non-blocking implementation of G-EDF to be useful in real-time systems, strong progress guarantees would be required, and ensuring such guarantees without greatly sacrificing efficiency is not easy.

Future Work. There are numerous directions for future work. First, we would like to evaluate the impact of dynamic workload changes, especially on the C-EDF variants, in order to give guidelines on the best cluster size to use in such contexts. Second, we would like to investigate the impact of non-preemptable critical sections and synchronization on schedulability. Third, we would like to investigate C-EDF approaches within mixed HRT/SRT contexts. Finally, we would like to tackle the problem of designing a non-blocking G-EDF scheduler.

References

[1] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. of the 24th IEEE Real-Time Sys. Symp.*, pp. 120–129, 2003.

[2] T. Baker. A comparison of global and partitioned EDF schedulability tests for multiprocessors. Technical Report TR-051101, Florida State University, 2005.

[3] T. Baker and S. Baruah. Schedulability analysis of multiprocessor sporadic task systems. In *Handbook of Real-Time and Embedded Sys.*. Chapman Hall/CRC, 2007.

[4] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proc. of the 28th IEEE Real-Time Sys. Symp.*, pp. 119–128, 2007.

[5] A. Bastoni, B. Brandenburg, and J. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor real-time schedulers. Extended version of this paper. Available at http:// www.cs.unc.edu/ ˜anderson/papers.html.

[6] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *Proc. of the 6th Int'l Workshop on Operating Sys. Platforms for Embedded Real-Time Apps.*, pp. 33–44, 2010.

[7] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proc. of the 17th Euromicro Conf. on Real-Time Sys.*, pp. 209–218, 2005.

[8] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Trans. on Parallel and Distributed Sys.*, 20(4):553–566, 2009.

[9] B. Brandenburg and J. Anderson. On the implementation of global real-time schedulers. In *Proc. of the 30th IEEE Real-Time Sys. Symp.*, pp. 214–224, 2009.

[10] B. Brandenburg and James H. Anderson. Feather-trace: A light-weight event tracing toolkit. In *In Proc. of the Third Int'l Workshop on Operating Sys. Platforms for Embedded Real-Time Apps.*, pp. 61–70, 2007.

[11] B. Brandenburg, J. Calandrino, and J. Anderson. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proc. of the 29th IEEE Real-Time Sys. Symp.*, pp. 157–169, 2008.

[12] B. Brandenburg, H. Leontyev, and J. Anderson. Accounting for interrupts in multiprocessor real-time systems. In *Proc. of the 15th IEEE Int'l Conf. on Embedded and Real-Time Computing Sys. and Apps.*, pp. 273–283, 2009.

[13] J. Calandrino, J. Anderson, and D. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proc. of the 19th Euromicro Conf. on Real-Time Sys.*, pp. 247–256, 2007.

[14] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. LITMUS$^{\mathrm{RT}}$: A testbed for empirically comparing real-time multiprocessor schedulers. In *Proc. of the 27th IEEE Real-Time Sys. Symp.*, pp. 111–123, 2006.

[15] U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, University of North Carolina, Chapel Hill, North Carolina, 2006.

[16] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Sys.*, 38(2):133–189, 2008.

[17] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Sys.*, 25(2-3):187–205, 2003.

[18] UNC Real-Time Group. LITMUS$^{\mathrm{RT}}$ homepage. http://www.cs.unc.edu/ ˜anderson/litmus-rt.

[19] Intel Corp. Intel Unveils New Product Plans for High- Performance Computing. http://www.intel.com/pressroom/archive/releases/2010/20100531comp.htm, 2010.

[20] F. Liu, F. Guo, Y. Solihin, S. Kim, and A. Eker. Characterizing and modeling the behavior of context switch misses. In *Proc. of the 17th Int'l Conf. on Parallel Architectures and Compilation Techniques*, pp. 91–101, 2008.

[21] J. Liu. *Real-Time Sys.*. Prentice Hall, 2000.

[22] P. McKenney. Shrinking slices: Looking at real time for Linux, PowerPC, and Cell. http://www.ibm.com/developerworks/power/library/pa-nl14-directions.html, 2005.

[23] NIST/SEMATECH. e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/, 2010.

[24] S. Schliecker, M. Negrean, and R. Ernst. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *Design, Automation Test in Europe Conf. Exhibition*, pp. 759–764, 2010.

[25] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proc. of the 20th Euromicro Conf. on Real-Time Sys.*, pp. 181–190, 2008.

[26] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. on Embedded Computing Sys.*, 7(3):1–53, 2008.

[27] J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *Proc. of the 14th IEEE Real-Time and Embedded Technology and Apps. Symp.*, pp. 80–89, 2008.