CHAPTER 7

Conclusion

Prior work has established that G-EDF is a useful scheduler for multiprocessor real-time systems in which bounded lateness is acceptable, and has described how to compute lateness bounds. In this dissertation, we have demonstrated that the broader class of GEL schedulers can be used and modified in order to provide improved performance in the form of smaller lateness bounds. Furthermore, prior work on the MC$^2$ scheduling framework has not accounted for overload caused by jobs running for longer than their level-C PWCETs. We have provided a mechanism to recover from such overloads, in addition to bounds on the time required for the system to return to normal in the absence of continued overload.

7.1 Summary of Results

In this dissertation, we have supported the thesis that *G-EDF can be modified to support smaller lateness bounds than previous work allows, with more flexibility to specify desired lateness criteria. Furthermore, such modifications do not violate the assumptions required for multiprocessor locking protocols, and the modified scheduler is easier to implement and/or has lower overheads than known HRT-optimal schedulers. In addition, recovery from overloads caused by tasks in MC$^2$ overrunning their level-C PWCETs can be facilitated by modifying the scheduler to delay job releases dynamically.*

To support this thesis, we have provided improved analysis of GEL schedulers and proposed modifications to G-EDF. We now describe these contributions.

**Compliant-Vector Analysis.** In Chapter 3, we have proposed CVA as an improved method to compute tardiness bounds under GEL schedulers. Compared to the initial work of Devi and Anderson (2008), we have provided more finely tuned bounds for different tasks within the same system and
allow PPs that differ from deadlines, resulting in response-time bounds of the form $Y_i + x_i + C_i$ instead of $D_i + x + C_i$. As a result of this change, response-time bounds may be less than relative deadlines. Under such circumstances, tardiness is simply zero, but lateness is negative. Therefore, we provide better bounds by considering lateness bounds rather than tardiness bounds. Lateness bounds can be computed using an LP.

**Global-Fair-Lateness Scheduling.** Also in Chapter 3, we have proposed G-FL as a scheduler that provably provides under CVA the best maximum lateness bound under a task system. This property is achieved by ensuring that all tasks in the system have the same lateness bounds. G-FL is simply defined with $Y_i = D_i - \frac{m-1}{m} C_i$, resulting in a simple implementation.

**LP Techniques for Assigning PPs.** Our final contribution in Chapter 3 has been to propose methods to use an LP to select GEL schedulers to achieve lateness properties that differ from those provided by G-FL. The same LP used to compute lateness bounds for a given CVA scheduler can be extended to include relative PPs as variables and linear functions of lateness as the objective function to be minimized. With such a technique we have shown, for example, that lateness bounds for some tasks can be further reduced from G-FL (although the maximum lateness bound in the system cannot be reduced) and that we can minimize the maximum lateness bound proportional to a task’s relative deadline.

**Removing Intra-Task Precedence Constraints.** In Chapter 4, we have shown that allowing multiple jobs of the same task to run in parallel can provide several benefits. When only one job of each task can run at a time, no task’s utilization can exceed one, or that task’s tardiness may grow without bound. However, by allowing multiple jobs of each task to run in parallel, it is only necessary that the entire system not be overutilized. Furthermore, even when allowing multiple jobs of the same task is not necessary for schedulability, doing so can result in smaller lateness bounds. We have provided a method to compute such bounds.

**Job Splitting.** In Chapter 5, we have considered a technique to split jobs in order to further reduce lateness bounds. With our technique, each job of a task is split into a per-task constant number of subjobs. It is not necessary for the developer to explicitly split the job in source code; rather, the operating system performs the splitting by enforcing subjob budgets. In the absence of overheads and critical sections, such splitting could ensure lateness bounds arbitrarily close to zero. We
have provided an experimental study, consisting of both an implementation and a schedulability component, demonstrating the improvements that can be obtained while considering overheads in a real system. We have also provided a method to handle critical sections that is compatible with existing locking protocols, and we have considered the effects of critical sections on achievable lateness bounds.

**General Analysis of Level C of MC².** In Chapter 6, we have provided improved analysis for level C of MC². While Leontyev and Anderson (2010) provided restricted supply analysis that can be applied in such a setting, we have provided three improvements. First, we have tightened the analysis in the case that all processors are mostly, but not fully, available to level C. Such a situation is the common case for MC². Second, we have provided analysis that is general enough to apply *even when some jobs exceed their level-C PWCETs*. Finally, our analysis explicitly accounts for the virtual time mechanism we describe in the next paragraph.

**Virtual Time at Level C of MC².** If an overload occurs in which jobs exceed their level-C PWCETs, it is possible that the system may never recover or may recover slowly, even in the absence of continued overload. In order to resolve this problem, we have also provided, in Chapter 6, a mechanism to recover from such overloads. Instead of defining job PPs and task minimum separation times with respect to the actual time, we define PPs and minimum separation times with respect to *virtual* time. In the absence of overload, virtual time and actual time are identical, but the system can recover from overload by causing virtual time to elapse more slowly than actual time. This reduces the effective amount of work at level C, assisting the system in recovering from the overload.

**Dissipation Time and Bounds at Level C of MC².** Additionally in Chapter 6, we have provided a condition that ensures that the system has “returned to normal,” and that virtual time can safely proceed at the same speed as actual time after this condition is satisfied. We have provided two forms of analysis of the dissipation time, or the time required for the system to reach this condition once an overload has completed. First, we have provided an experimental study of the dissipation time on a real system. Second, we have provided theoretical dissipation bounds, or bounds on dissipation time.
7.2 Other Related Work

In this section, we briefly describe our other contributions to the field of real-time systems that have not been included in this dissertation.

**Original MC\(^2\) Proposal.** In (Mollison et al., 2010), we proposed MC\(^2\) in its original form with five criticality levels. In that form, an additional SRT level D is also present, is scheduled under G-EDF, and is prioritized below level C. Otherwise, that form of MC\(^2\) is identical to that reviewed in Chapter 1. In this dissertation, we have focused on the newer MC\(^2\) model and have considered only level C.

**Scheduling in Google Earth.** In (Erickson et al., 2012), we considered scheduling within the virtual globe software Google Earth. We considered scheduling techniques to reduce the problem of *stutter*, where the display cannot update to a new image at the correct time, preventing smooth animation.

This work involved predicting how long certain jobs within Google Earth would run. However, because Google Earth runs on a wide variety of software and hardware platforms, it is not possible to determine job execution times *a priori*. Thus, we proposed a system to estimate job execution times online. We also implemented our system in Google Earth, and it has been included in the official Google Earth release since version 6.2.

**Scheduling Automotive-Inspired Dataflows.** In (Elliott et al., 2014), we considered the multicore scheduling of DAG-based systems with producer/consumer dependencies. In such systems, dependencies are specified using a DAG, and each job must wait to start until all its corresponding jobs in its predecessor tasks have finished. We extended prior work (Liu and Anderson, 2010, 2011) by adding additional constraints to reduce pessimism, using C-FL (the clustered variant of G-FL) in place of C-EDF (the clustered variant of G-EDF), using the job splitting technique proposed in Chapter 5, and intelligently assigning tasks to clusters of processors to reduce cache-related overhead. We performed an implementation study to analyze response times.

**Optimal Soft Real-Time Semi-Partitioned Scheduling.** In (Anderson et al., 2014), we proposed the *EDF-os* algorithm for SRT scheduling. EDF-os is derived from EDF-fm (Anderson et al., 2008), which is a *semi-partitioned* scheduler that does not allow most tasks to migrate between processors, but allows certain tasks to migrate at job boundaries. EDF-fm required restrictions on
per-task utilizations. In EDF-os, we provided several modifications that eliminated the need for such restrictions. We also provided both lateness bounds for EDF-os and an implementation study comparing EDF-os to several other algorithms. EDF-os can often provide smaller lateness bounds than G-FL, at the expense of not allowing the task system to be modified at runtime.

7.3 Future Work

We now discuss some future work that could improve the results presented in this dissertation.

Tighter Lateness Bounds. While the lateness bounds provided in Chapter 3 are tighter than those provided in previous work, we still do not believe that they are tight. There are two pessimistic assumptions that may be able to be lifted. For one, we assume that jobs do not begin execution before their PPs, in order to avoid some complex edge cases in our analysis. Additionally, we assume that each task that carries work into the beginning of a busy interval has as much lateness as possible. It may be possible to prove that this cannot be the case.

Selection of GEL Schedulers in the Absence of Intra-Task Precedence Constraints. As discussed in Chapter 4, the LP techniques used in Chapter 3 cannot immediately be applied in the absence of intra-task precedence constraints. Therefore, although our analysis is general enough to apply to arbitrary GEL schedulers, we do not specify either a scheduler like G-FL that provably has particular desired properties, or a method to choose a GEL scheduler to minimize particular lateness criteria. The same insights used to compute the response-time bounds in Chapter 4 may also be useful to achieve these goals using a method other than an LP.

Splitting Jobs with Other Locking Analysis. In Chapter 5, we consider only one mutex queue spin-based locking, where a CPU waiting to acquire a lock consumes CPU until the lock is available. However, the technique we use could also be applied to other forms of locking protocols. We leave such extensions as future work.

More Flexible Mechanisms for Overload Management at Level C in MC². The virtual time mechanism proposed in Chapter 6 equally penalizes all level-C tasks when recovering from overload. This technique makes sense in a mixed-criticality setting, because tasks of greater importance than others could be moved to level A or level B. However, a finer-grained notion of importance may be
preferable: some tasks may not require the full rigor of level-A or level-B analysis, but may be more important in an overload than other level-C tasks. Alternative mechanisms that allow different tasks to be treated differently would be useful in such situations.