Utility Accrual Scheduling of Distributable Threads: The *Tempus* Approach

Peng Li, Binoy Ravindran
Real-Time Systems Laboratory
Virginia Tech, Blacksburg, VA 24061, USA
{peli2,binoy}@vt.edu

E. Douglas Jensen
The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

1. Introduction

Dynamic, adaptive, real-time embedded control systems are present at any level(s) of an enterprise—e.g., devices in the defense domain such as multi-mode phased array radars and battle management. These embedded systems often include "soft" as well as "hard" time constraints.

Jensen's time/utility functions [4] (or TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF specifies the utility to the system, resulting from the completion of an activity, as a function of the its completion time. Figure 1 shows examples of TUF time constraints. TUFs have been successfully used in two significant, real-time applications, including an AWACS (Airborne WArning and Control System) surveillance mode tracker system built by MITRE and Open Group, and a coastal air defense system built by CMU and General Dynamics.

When timing constraints are expressed with TUFs, the scheduling optimality criteria are based on factors in terms of maximizing accrued utility from those activities—e.g., maximizing the sum, or the expected sum, of the activities' attained utilities. Such criteria are called *Utility Accrual* (or UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms.

Motivated by the need for dynamic scheduling, e.g., UA scheduling, in real-time distributed systems, the Object Management Group recently adopted the Real-Time CORBA 2.0 (Dynamic Scheduling) standard [7] (abbreviated here as RTC2[1]). RTC2 specifies *distributable threads* (or *DT*s) as a programming and scheduling abstraction for system-wide, end-to-end scheduling in real-time distributed systems. The distributable thread model is a subset of the *distributed thread* model that was created in Jensen's Archons Project [3].

A DT is a single thread of execution with a globally unique identifier that transparently extends and retracts through an arbitrary number of local and remote objects. Concurrency is at the DT-level.

1 Real-Time CORBA 2.0 has been recently renamed Real-Time CORBA 1.2.

A DT carries its execution context as it transits node boundaries, including information such as the thread's scheduling parameters (e.g., time constraints, execution time, importance), identity, and security credentials.

We have developed the *Tempus* middleware that supports DTs as a programming and scheduling abstraction for system-wide, end-to-end scheduling [6]. DTs in *Tempus* can be subject to time constraints including those specified using arbitrarily-shaped TUFs and timeliness optimality criteria including maximizing accrued utility. In the rest of this paper, we overview the *Tempus* middleware.

2. The *Tempus* Middleware

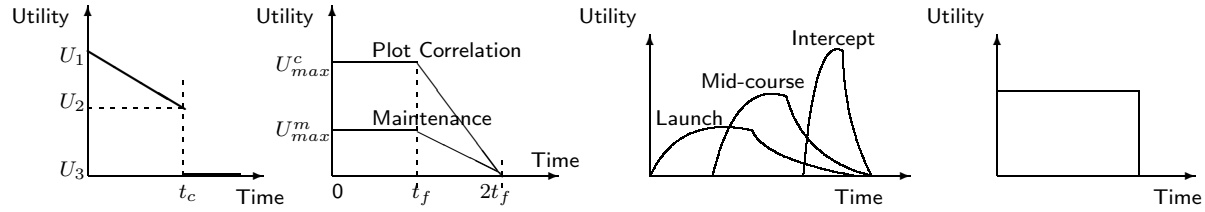
Core components of *Tempus* include an application-level UA scheduling framework, a portable interceptor, and a node-local UA scheduling algorithm.

Tempus' application-level UA scheduling framework, called *meta-scheduler* provides a mechanism for implementing UA scheduling algorithms on top of POSIX RTOSes. It exclusively uses real-time POSIX APIs and thus enjoys good portability.

Since *Tempus* exclusively uses POSIX RTOSes as its underlying base, the abstraction of DTs needs to be implemented as native OS abstractions, such as processes and threads. Each node in a real-time distributed systems runs an instance of the portable interceptor component. A portable interceptor (PI) is responsible for mapping a DT to a native OS thread and maintains that mapping information while the DT has a segment (active or blocked) on the PI's residing node.

Scheduling of DTs in *Tempus* is performed according to RTC2's Case 2 approach, i.e., local scheduling on each node using propagated timeliness parameters. Thus, the scheduler instance on each node resolves resource dependencies and constructs local schedules in a way that seeks to maximize locally accrued utility and thus obtain approximate, globally optimal accrued utility.

A DT may contain one or more potentially nested *scheduling segments*. A scheduling segment delimits the



(a) Track Asso. (b) Corrln. & Maint. (c) Missile Control (d) A Step TUF

Figure 1. Example Time Constraints Specified Using TUFs

part of a DT's control flow that is subject to the specified time constraints.

The scheduling parameters in *Tempus* are only meaningful to the node schedulers, and thus can have arbitrary format as long as the node schedulers can interpret them properly. In the current version of *Tempus*, the set of scheduling parameters must first specify the scheduler to be invoked. *Tempus* currently implements eight schedulers, including fixed priority schedulers, deadline-driven schedulers such as the Earliest Deadline First (EDF) scheduler, the Dependent Activity Scheduling Algorithm (DASA) scheduler [1], and the Generic Utility Scheduling (GUS) scheduler [5]. In addition, scheduling information associated with the designated scheduler needs to be specified, such as priorities for the fixed priority scheduler and deadlines for the EDF scheduler.

3. Experimental Results

Our test bed contains a network of PCs running RedHat Linux and QNX Neutrino operating systems. The experiments use periodically spawned DTs with downward-step TUF, decreasing TUF, and parabolic TUF time constraints. We conducted experiments using four schedulers, including a RMA scheduler, an EDF scheduler, a DASA scheduler, and a GUS scheduler.

Table 1 shows the mean's and standard deviations of the Accrued Utility Ratio (AUR) under the four schedulers. AUR is the ratio of accrued utility to the maximal possible utility of all DTs. We observe that (1) the standard deviations of all performance measurements are small regardless of the mean values, which suggest that the performance of *Tempus* is stable and predictable; and (2) the GUS scheduler outperforms all other schedulers during "low load" as well "high load."

Load	Algorithm	Avg(AUR)	Std(AUR)
Lowload	**RMA**	77.262	1.643706
	EDF	76.5	0.453045
	DASA	77.248	2.123127
	GUS	87.604	0.571384
Highload	**RMA**	64.202	1.105812
	EDF	66.662	2.83841
	DASA	67.474	2.29438
	GUS	80.16	2.146404

Table 1. Performance of Scheduling Policies

4. Conclusions

The *Tempus* middleware thus illustrates the effectiveness of scheduling DTs using propagated scheduling parameters (including arbitrarily-shaped TUFs) for node-local scheduling and resource contention resolution. Ongoing efforts include incorporating TMAR (thread maintenance and repair) protocols [2] into *Tempus* and developing scheduling algorithms for DTs that use RTC2's Case 3 and Case 4 approaches.

References

[1] R. K. Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, 1990.

[2] J. Goldberg, I. Greenberg, R. K. Clark, E. D. Jensen, K. Kim, and D. M. Wells. Adaptive fault-resistant systems. Technical Report csl-95-02, Computer Science Laboratory, SRI International, Menlo Park, CA., January 1995. `http://www.csl.sri.com/papers/sri-csl-95-02/`.

[3] E. D. Jensen. The Archons project: An overview. In *Proceedings of the International Symposium on Synchronization, Control, and Communication*. Academic Press, 1983.

[4] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time systems. In *IEEE RTSS*, pages 112–122, December 1985.

[5] P. Li. *A Utility Accrual Scheduling Algorithm for Resource-Constrained Real-Time Activities*. Phd dissertation proposal, Virginia Tech, 2003. Available at `http://www.ee.vt.edu/~realtime/li-proposal03.pdf`.

[6] P. Li, H. Cho, B. Ravindran, and E. D. Jensen. Scheduling distributable real-time threads in Tempus middleware. In *IEEE Intl' Conf. on Parallel and Dist. Systems*, 2004. to appear.

[7] OMG. Real-time CORBA 2.0: Dynamic scheduling specification. Technical report, Object Management Group, September 2001. OMG Final Adopted Specification, `http://www.omg.org/docs/ptc/01-08-34.pdf`.