Approved for Public Release; Distribution Unlimited
Case #04-00411

Time/Utility Function Decomposition in Soft Real-Time Distributed Systems

Abstract

We consider Real-Time CORBA 2.0 (Dynamic Scheduling) *distributable threads*, whose time constraints are specified using time/utility functions (TUFs), operating in *legacy environments*. In legacy environments, system node resources—both physical (processor, disk, I/O, etc.) and logical (locks, etc.)—are shared among time-critical distributable threads and local applications that may or may not be time-critical. Thus, in such environments, distributable threads that are scheduled using their propagated TUFs and scheduling parameters, as mandated by Real-Time CORBA 2.0's Case 2 approach, may suffer performance degradation, if a node scheduler can achieve higher local accrued utility by giving higher eligibility to local threads than to distributable threads. To alleviate this, we consider decomposing TUFs of distributable threads into "sub-TUFs" that are used for scheduling segments of distributable threads. We present methods for decomposing TUFs. Furthermore, we identify conditions under which TUF decomposition can alleviate performance degradation. Our experimental results reveal that the most important factors that affect the performance of TUF decomposition include the properties of node scheduling algorithms, TUF shapes, *task load*, *Global Slack Factor*, local threads and resource dependencies, and that these factors interact.

Index Terms

real-time distributed systems, time/utility functions, real-time scheduling, soft real-time systems, time constraint decomposition, real-time CORBA

I. Introduction

The Object Management Group's recently adopted Real-Time CORBA 2.0 (Dynamic Scheduling) standard [1] (abbreviated here as RTC2[1]) specifies *distributable threads* (or *DT*s) as a programming and scheduling abstraction for system-wide, end-to-end scheduling

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

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2006		**00-00-2006 to 00-00-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Time/Utility Function Decomposition in Soft Real-Time Distributed Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **MITRE Corporation,202 Burlington Road,Bedford,MA,01730-1420**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**42**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

in real-time distributed systems. 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. A DT is thus an end-to-end control flow abstraction, with a logically distinct locus of control flow movement within/among objects and nodes. Concurrency is at the DT-level. Thus, a DT always has a single execution point that will execute at a node when it becomes "most eligible" as deemed by the node scheduler. 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. Hence, DTs require that Real-Time CORBA's *Client Propagated* model be used, not the *Server Declared* model. Figure 1 cited from [1] shows the execution of DTs.

The propagated parameters are used by the schedulers on each of the nodes the DT transits, for resolving all node-local resource contentions among DTs and for scheduling DTs on nodes to satisfy the system's timeliness optimality. Using the same optimality criterion with the same parameters on each node that a DT transits results in approximate, system-wide timeliness optimality. This distributed scheduling approach, called *Distributed Scheduling: Case 2* in the RTC2 specification, is explicitly supported in RTC2 due to its simplicity and capability for coherent end-to-end scheduling.[2]

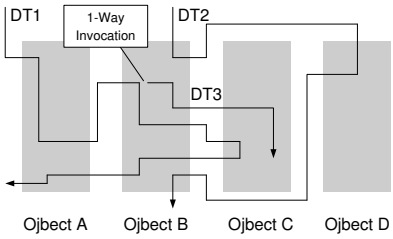


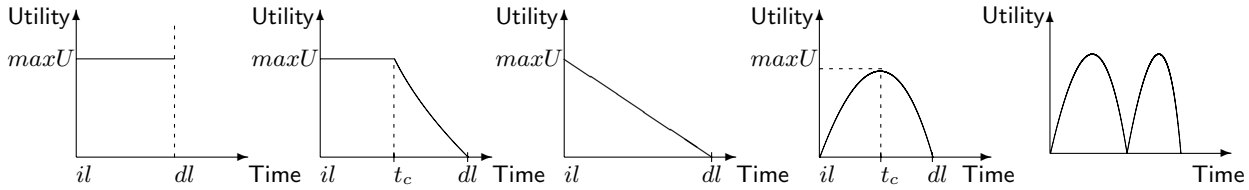
Fig. 1. Distributable Threads

In this paper, we focus on complex, dynamic, adaptive real-time systems at any level(s) of an enterprize—e.g., in the defense domain, from devices such as multi-mode phased array radars [2] to battle management [3]. Such systems include "soft" as well as hard time constraints in the sense that completing a time-constrained activity at any time will result in some utility to the system, which depends on the activity's completion time. Such soft real-time constraints may be as important or mission-critical as hard deadlines.

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 that results from the com-

[2]RTC2 also describes Cases 1, 3, and 4, which describe non real-time, global, and multilevel distributed scheduling, respectively [1]. However, RTC2 does not support Cases 3 and 4.

pletion of an activity as a function of its completion time. Figure 2 shows the conventional deadline (downward step) and several soft time constraints specified using TUFs.



(a) Step (b) Soft-Step (c) Linear (d) Parabolic (e) Multimodal

Fig. 2. Deadline and Example Soft Timing Constraints Specified Using Jensen's Time/Utility Functions

When time constraints are expressed with TUFs, the scheduling optimality criteria are based on factors that are 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 (UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms. In general, other factors may also be included in the optimality criteria, such as resource dependencies and precedence constraints. Several UA scheduling algorithms are presented in the literature [5]–[11]. RTC2 has IDL interfaces for the UA scheduling discipline, besides others such as fixed-priority, earliest deadline first, and least laxity first.

Thus, according to RTC2's Case 2 approach, DTs whose time constraints are expressed using TUFs can be scheduled using their propagated TUFs and scheduling parameters. The propagated TUFs can be used by node-local UA scheduling algorithms to resolve local resource contentions and to construct local schedules that maximize *locally* accrued utility and approximate *global* accrued utility.

We consider integrating RTC2 applications, whose DT time constraints are specified using TUFs, into "legacy" environments. In such cases, a system node's physical (processor, disk, I/O, etc.) and logical (locks, etc.) resources are shared between one or more RTC2 applications and other non-RTC2 applications, some of which include threads having TUF time constraints. We refer to such threads as "local threads." Real-Time CORBA 1.0 (RTC1) [12] has an analogous legacy issue, in that any nodes may be shared by both RTC1 applications and non-CORBA applications. RTC1 priorities are mapped into each node's local priority

space, which is shared by RTC1 and non-CORBA applications.

In legacy environments, resource-contention resolution and scheduling of DTs using their propagated TUFs may not always be the best approach. For example, local threads in an application on a node may always be favored by that node UA scheduler, at the expense of the DTs of RTC2 applications, because the node scheduler may find that favoring the local threads leads to higher *locally* accrued utility (due to the particular shape of the TUFs of local threads). However, higher local utility does not necessarily imply higher system-wide utility in terms of the sum of utilities attained by all DTs, which is our optimization objective. Thus, DTs of an RTC2 application can suffer interference from local threads, causing them to perform poorly.

Besides the shape of TUFs, a number of other factors may also affect the performance of DTs in legacy environments. Example factors include the mixture of local threads and DTs, laxity of those local threads with deadlines, execution times of DTs and local threads, and the UA scheduling algorithms employed at all the nodes.

To help DTs properly compete with local threads and improve their performance in legacy environments, their TUFs can be *decomposed*. We hypothesize that it may be possible to decompose the TUF of a DT into "sub-TUFs" for each segment of the DT, so that the sub-TUFs can be used for node-local resource-contention resolution and UA scheduling. This hypothesis raises fundamental questions such as "how to decompose TUFs, both step downward shaped and non-step shaped?" Furthermore, "under what conditions can TUF decomposition help DTs improve their performance in legacy environments?"

In this paper, we answer these questions. We identify the conditions under which RTC2 DTs suffer performance degradation in an legacy environment. Furthermore, we present methods for decomposing TUFs and identify conditions under which TUF decomposition can improve DT performance. Through extensive simulations, we show that although TUFs of DTs can be decomposed to improve their performance in legacy environments, the performance of TUF decomposition is affected by many factors, among which the most important ones include the properties of node-local schedulers, TUF shapes, system load, and local threads. These factors interact with each other, and their effects on the performance of

TUF decomposition and working conditions are summarized at the end of this paper. We are not aware of any other efforts that have studied TUF decomposition (though deadline-decomposition has been studied).

TUF decomposition is analogous to the problem of deadline decomposition in distributed real-time systems whose time constraints are deadlines, and to the problem of priority mapping in RTC1. In all three cases, there are end-to-end timeliness requirements that must be decomposed into timeliness requirements on each node involved in a multi-node computation. On those nodes, these decomposed timeliness requirements contend for resources with the timeliness requirements of strictly node-local computations. The challenge is for the scheduler to resolve this contention in a manner that is optimal according to application-specific criteria for both the distributed computations and the local computations. A multi-node computation with an end-to-end deadline has local sub-computations with per-node deadlines derived from the end-to-end deadline; these deadlines contend with the node-local computation deadlines—the prior work on this topic is summarized in Section VII. In RTC1 systems, an RTC1 thread has a 15-bit CORBA ("global") priority; when a client invokes a servant, the client CORBA priority is mapped into the servant node's local operating system (much smaller) priority space using an application- or system-specific mapping; these priorities contend with the node-local computation priorities. (The mapping is reversed when an invocation returns; care must be taken in defining the mappings so as to retain priority fidelity end-to-end.)

The rest of the paper is organized as follows: In Section II, we provide motivation for our TUF and UA model by summarizing two significant demonstration applications that were successfully implemented using that model. In Section III we describe the thread and system model to study the TUF decomposition problem. Section IV and Section V lists the possible factors affecting TUF decomposition and the decomposition strategies we proposed for this problem, respectively. In Section VI, we describe our experimental setup, experimental evaluation, and analyze the results. In Section VII, we overview past research on time constraint decomposition in real-time distributed systems and contrast with our work. Finally, the paper concludes and identifies future work in Section VIII.

II. Motivating Application Examples for TUFs

As example real-time systems requiring the expressiveness and adaptability of TUF time constraints, we summarize TUFs of two applications. These include: (1) AWACS (Airborne WArning and Control System) surveillance mode tracker system [13] built by The MITRE Corporation and The Open Group (TOG); and (2) a coastal air defense system [14] built by General Dynamics (GD) and Carnegie Mellon University (CMU). We only summarize some of the application time constraints here; other details can be found in [13], [14], respectively.

A. TUFs in AWACS

The AWACS is an airborne radar system with many missions, including air surveillance. Surveillance missions generate aircraft tracks for command and control (C2) and battle management (BM). The surveillance tracker consists of several different activities. Its most demanding computation, called *association*, associates sensor reports to aircraft tracks. The tracker employs two sensors that sweep 180 degrees out of phase with a ten second period. Thus, association has a "critical time" at the period length. If the computation can process a sensor report for a track in under five seconds (half the sweep), that will provide better data for the corresponding report from the out-of-phase sensor. Thus, prior to critical time, utility of association decreases as critical time nears.

After the critical time, the utility of association is zero, because newer sensor data has probably arrived. Thus, if the processing load in one sensor sweep period is so heavy that it cannot be completed, probably the load will be about the same in the next period. So there will not be any resources to also process sensor data from the previous sweep.

This timeliness behavior, which requires the expressiveness and adaptability of soft yet mission-critical time constraints, would be difficult to describe using priorities. An effective solution is to describe it using TUFs.

The described semantics establish association's TUF shape: a critical time t_c at the sweep period; utility that decreases from a value U_1 to a value U_2 until t_c; and an utility value U_3 after t_c. U_1, U_2, and U_3 are determined using Application

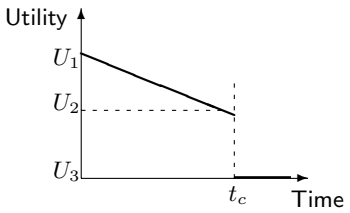
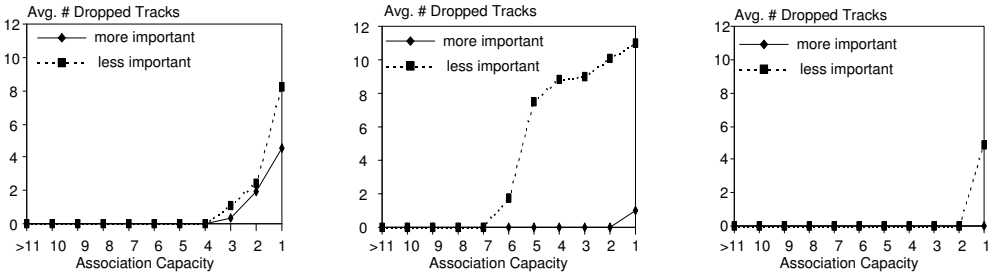


Fig. 3. Track Association TUF

QoS (AQoS) metrics such as: (1) track quality, which is a measure of the amount of sensor data incorporated in a track record; (2) track accuracy, which is a measure of the uncertainty in the estimate of a track's position and velocity; and (3) track importance, which is measure of track attributes such as its threat. Figure 3 shows the association thread's TUF.

The tracker creates threads for each airborne object that it tracks. The threads perform a sequence of activities, including association. The TUFs of all threads have the same basic shape shown in Figure 3, but use different values for U_1, U_2, and U_3. The system's UA scheduling algorithm resolves the resource contention among all the association (and other) threads and schedules system resources to maximize the total summed utility.



(a) FIFO (b) Fixed Priority (c) UA

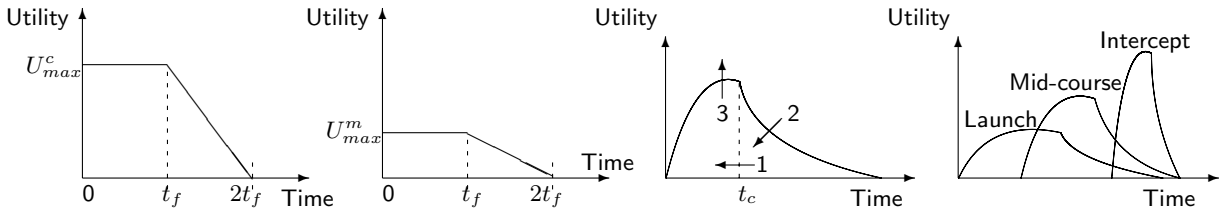
Fig. 4. Average Number of Dropped Tracks vs. Association Capacity

The AWACS surveillance tracker implementation was done using TOG's MK7 operating system [15], [16]. MK7 contains the UA scheduling algorithm described in [6]. To understand how well MK7's UA algorithm is able to schedule system resources in a mission-oriented way, significant performance measurements were made. Different scheduling algorithms, including FIFO and fixed priority, were compared with [6]. Figure 4 shows the average number of dropped tracks for the three scheduling policies under decreasing association capacity. The figure illustrates that the UA algorithm minimizes the number of dropped tracks, thereby illustrating the adaptivity of the TUF/UA paradigm.

B. TUFs in Coastal Air Defense System

The coastal air defense system defends the coastline from incoming cruise missiles and bombers, using a variety of assets including guided interceptor missiles. Time constraints

of three activities in the GD/CMU coastal air defense system, called *plot correlation*, *track maintenance*, and *missile control* are shown in Figures 5(a), 5(b), and 5(c), respectively.



(a) Plot Correlation (b) Track Maintenance (c) Missile Control (d) Missile Ctrl. Shapes

Fig. 5. TUFs of Three Activities in GD/CMU Coastal Air Defense

AQoS metrics such as track quality and weapon spherical error probable are used to define how each service's timeliness contributes to its utility to the current state of the mission. Note that the TUF's for sending guidance updates to the interceptor missiles have shapes that evolve during the course of each missile's engagement with its incoming target. This adaptive effect is extremely difficult to achieve with priorities. Performance evaluation of the system proves the effectiveness of TUF/UA. For brevity, here we skip the details of TUF, application implementation and experimental characterizations measuring adaptive timeliness; these can be found in [14]. (The application architecture that uses DTs is shown in Figure 6.)

III. The Application, Timeliness, and System Models

A. The Thread Model

We assume that the application consists of a set of DTs and local threads. A DT can execute in objects that are distributed across computing nodes by location-independent invocations and returns. Within each node, the flow of control is equivalent to normal local thread execution. One possible implementation is to map a DT to a local thread while it is executing in each node. We will refer to each node-local segment of a DT as an object-level thread (or simply as OLT) hereafter. Therefore, a DT can be assumed to consist of a series of OLTs that the DT is mapped to along its execution path.

For achieving fault-tolerance, it is possible that application objects may be replicated.

Thus, when a DT invokes a method on a replicated object, multiple OLTs will execute as part of the method executions on the object replicas. When an OLT in a replicated object completes its execution, control is transferred to a synchronization point, which is usually the next object method in the invocation chain of the DT.

We denote the set of DTs by $\mathbf{T} = \{DT_k : 1 \leq k \leq n\}$. As a shorthand, we use the notation $DT_k = \{OLT^k_{i,j} : 1 \leq i \leq m_k, j \geq 1\}$ to represent the kth DT that consists of m_k OLTs, where $OLT^k_{i,j}$ means the jth replica of the DT's i^{th} segment. The variable m_k is also called the number of segments of DT_k.

We assume that the locus of control flow movement of the DTs are known. Thus, the chain of method invocations of each DT is assumed to be a-priori known. For many applications, it is possible to obtain this knowledge by static code analysis [17]. Of course, for some applications, such static code analysis will be difficult and hence it will be difficult to a-priori know the DT chain of method invocations.

The GD/CMU air defense system described in [14] is an example application that is implemented with DTs. Figure 6 illustrates the control flows of DTs which must be scheduled in this system, and different nodes representing the BM/C2 functions.

B. The System Model

We consider a system model, where a set of processing components, generically referred to as *nodes*, are interconnected via a communication network. Each node in the system executes OLTs of DTs as well as local threads generated at the node. The order of executing the threads—OLTs and local threads—on a node is determined by the scheduler that resides at the node. We assume that the node schedulers are completely independent of each other and do not collaborate, as described in RTC2's Case 2 approach. Thus, thread scheduling decisions made by a node scheduler are completely independent of other node schedulers.

Scheduling decisions made by the node schedulers are determined by the thread scheduling parameters, which typically include thread time constraints such as a deadline or TUF, importance, and the remaining execution time.

Nodes are assumed to be homogeneous in the sense that they have identical hardware

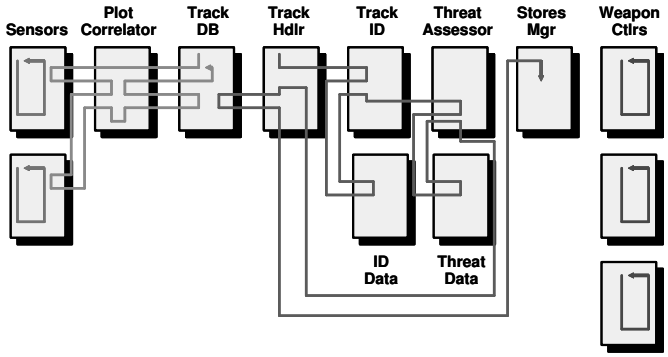


Fig. 6. DTs in GD/CMU Coastal Air Defense

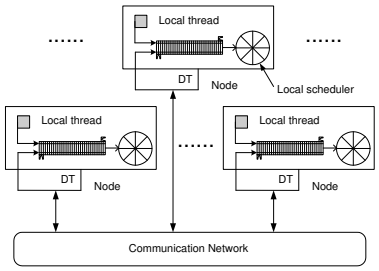


Fig. 7. The Distributed System Architecture

configurations in terms of processing speed, instruction pipeline, and primary memory/cache resources. Furthermore, all nodes are assumed to be running the same scheduling algorithm as the node scheduler. Figure 7 shows the distributed system architecture model.

C. Timeliness Model

We use TUF to specify the time constraint of a DT, an OLT, or a local thread, generically referred to as T, and denote the TUF of a thread T as $U(T)$. Thus, the completion of T at a time t will yield a timeliness utility $U\,(T, t)$. DTs propagate their TUFs as they transit nodes, and we decompose the propagated TUF of a DT into sub-TUFs for node-local scheduling.

TUFs can be classified into unimodal and multimodal functions. Unimodal TUFs are those TUFs for which any decrease in utility cannot be followed by an increase in utility. TUFs which are not unimodal are multimodal.

Example unimodal TUFs are shown in Figures 2(a), 2(b), 2(c) and 2(d). Note that the traditional soft deadline time constraint can be expressed as a "step downward" function, such as the one shown in Figure 2(a), where the completion of a thread at anytime before a certain time will result in uniform utility; completion of the thread after that time will result in zero utility. Example multimodal TUF is shown in Figure 2(e). We focus on unimodal TUFs in this paper, since they are used to specify a broad range of time constraints,.

Each TUF $U(T)$ is assumed to have an initial time $il(T)$ and a deadline time $dl(T)$. Initial time is the earliest time for which the function is defined, while deadline time is the latest

time at which the function drops to a zero utility value. Within this paper, we simply assume that $U(T)$ is defined from $il(T)$ until the future indefinitely. Thus, the term "deadline time" (or "deadline" in abbreviation) is used to denote the last point that a TUF crosses the *t-axis*. For a step downward TUF, the deadline time is its discontinuity point. Furthermore, we also assume that $U(T,t) \geqslant 0, \forall t \in [il(T), dl(T)]$ and $U(T,t) = 0, \forall t \geq dl(T)$.

We denote the arrival time of a thread T as $ar(T)$. The arrival time is simply the time at which the thread becomes ready for execution. The arrival time of a DT is the arrival time of the first OLT of the DT. We assume that $ar(T) = il(T)$. Hereafter, we use $ar(T)$ to represent both $ar(T)$ and $il(T)$.

The execution time of a DT/OLT/local thread T is denoted as $ex(T)$. The execution time of a DT is the sum of the execution times of all OLTs and transmission delays of all inter-OLT messages of the DT. Thus, a DT's execution time simply denotes the total workload of the DT.

IV. Factors Affecting TUF Decomposition

We list the factors that can influence TUF decomposition, potentially affecting performance of DTs, as follows:

1) **Scheduling algorithm.** The type of the scheduling algorithm employed at a node can impact TUF decomposition and thus DT performance. We consider UA scheduling algorithms such as GUS [5], DASA [7], LBESA [6] and D^{over} [8], and non-UA algorithms such as EDF [18] and LLF [19]. These algorithms make scheduling decisions that are based on different metrics. For example, the key concept for gaining accrued utility in GUS is a metric called *Potential Utility Density* (PUD) that was originally presented in [7]. LBESA and DASA uses PUD as a key decision-metric, but they also consider thread deadlines in computing their scheduling decisions. Furthermore, D^{over} is a timer-based UA algorithm. GUS, DASA and LBESA have the best performance among existing UA algorithms while D^{over} is optimal for some cases. The different characteristics of the scheduling algorithms will very likely affect the effects of TUF decomposition on DT performance.

2) **TUF shape.** The shapes of TUFs can affect TUF decomposition and thread schedul-

ing. For example, it would be beneficial to schedule a thread with a decreasing TUF as early as possible to accrue more utility, but for a thread with a strictly concave TUF, the scheduler may need to postpone its execution so as to complete it at the time corresponding to the optimum value of its TUF. In this paper, we only consider unimodal TUFs such as step downward TUFs, linear-shaped TUFs, and parabolic-shaped TUFs. Furthermore, different scheduling algorithms focus on TUFs with different shapes.

3) **Task load.** We define *task load* (or *load*, for short) as the ratio of the rate of work generated to the total processing capacity of the system. The analytical expression for load is given in Section VI-A starting from Page 18. Different scheduling algorithms produce different behaviors under different load. For example, The EDF algorithm is optimal during under-load situations in terms of satisfying all deadlines [18], but suffers domino effects during over-load situations. Some overload scheduling algorithms such DASA, mimics EDF to reap its optimality during under-loads, but provides much better performance during over-loads than EDF. However, GUS does not mimic the deadline-based scheduling algorithm such as EDF during under-loads, and it yields different timeliness utility during over-loads from EDF and DASA. Thus, load can affect TUF decomposition and DT performance.

4) **Global Slack Factor (GSF).** We define the *Global Slack Factor* of a DT as the ratio of the DT's execution time to its TUF's definition period, which is the sum of the DT's execution time and its slack. As an example, Figure 8 shows the slack and deadline time of a DT with number of segments $m = 4$. Therefore, the DT's GSF which describes its global stack is $GSF(T) = \frac{ex(T)}{dl(T)-ar(T)}$. Intuitively, the larger $GSF(T)$, the less global slack of T, which means more stringent time constraint is imposed on T, and it is more prone to complete after its deadline and accrue zero utility. Thus, in the TUF decomposition process, the GSF of a DT should be considered to change the relative importance of an OLT to compete with local threads.

5) **Homogenous/heterogeneous TUFs.** Each DT's end-to-end time constraint could possibly be specified using different TUFs. This can lead to DTs having homogenous or heterogenous TUFs. As different TUF shapes can affect the performance of decomposition, it is possible that a decomposition strategy can achieve improvement on one kind of TUFs

while deteriate the performance of another kind of TUFs at the same time. Thus, the TUF homogeneity/heterogenity can affect TUF decomposition and DT performance and should be studied.

6) **Dependencies.** It is possible for OLTs of DTs and local threads to have dependencies. Dependencies include resource access constraints such as mutual exclusion constraints and inter-thread precedence relationships. Some scheduling algorithms allow dependencies (e.g., [5], [7]), while others do not (e.g., [6], [8]). Dependencies bring more interference to the contentions among OLTs and local threads, with which a thread T is more likely to miss its deadline. Thus, TUF decomposition with such interference is more difficult to improve the DTs' performance.

7) **Local threads.** On each node, the OLTs of DTs compete with each other, and they also compete with the local threads. In addition, there also exists the contention among local threads. The three types of contentions, which we describe as global-global, global-local, and local-local, should be resolved by the node-local schedulers. Even though we do not decompose the TUFs of local threads, their properties will affect two types of contention, i.e., global-local and local-local, which can in turn affect the performance of DTs whose TUFs are decomposed and allocated to their OLTs.

V. TUF Decomposition Methods

A. Ultimate TUF

Without any specific knowledge on the execution times of the OLTs, the only available measure of their time requirement is the deadline and shape of their DT's TUF. Thus, a simple strategy would be to set the deadline time of an OLT to be equal to the deadline time of its DT, i.e., $dl(OLT_{i,j}^k) = dl(DT_k), 1 \leq i \leq m_k, j \geq 1\}$, and to set $U(OLT_{i,j}^k) = U(DT_k)$. We call this strategy as **Ultimate TUF (UT)**.

Thus, the *UT* strategy does not decompose a DT's TUF. The propagated (ultimate) TUF is used by all node schedulers for scheduling OLTs of a DT.

B. Slicing based on EQF

A problem with UT is that the time for the execution of a later OLT of a DT is considered slack to an earlier OLT. This may give the scheduler incorrect information about how much time an OLT can be delayed in its execution.

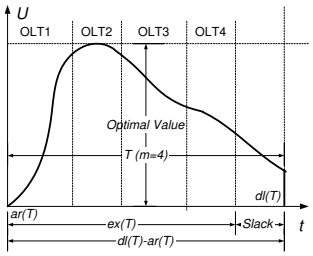


Fig. 8. The Global Slack of TUF

Thus, it is possible to "slice" the TUF of a DT by changing (only) the deadline time for each OLT of the DT. Such a slicing of the TUF based on the deadline time can be done using the Equal Flexibility (or EQF) strategy presented in [20]. EQF decomposes the end-to-end deadline of a global task (a DT in our case) into deadlines for subtasks (OLTs in our case), by dividing the total remaining slack among the OLTs in proportion to their execution times. The decomposition is done such that higher the execution time of a subtask, longer is its deadline. Thus, we can use EQF to decompose the deadline time of the TUF of a DT into non-overlapping thread execution windows (slices). We call this strategy **Slice on EQF (SLEQF)**, and it is illustrated in Figure 9 with an example DT of 3 segments.

$SLEQF$ first derives the deadlines of the OLTs of a DT from the DT's TUF using EQF. The TUF of an OLT is then defined as the segment of the TUF of the DT between the OLT's arrival time and its deadline.

C. Slicing based on TUF Shape

It is also possible to slice the TUF of a DT based on the TUF shape. We define the time at which the TUF reaches its extremum as Opt (for unimodal TUFs that we consider here, the extremum is the maximum). The utility value that corresponds to the Opt time is denoted $OptValue$. One intuition that we adopt in TUF decomposition is that, a DT should complete near its Opt to accrue as much utility as possible. Thus, we modify $SLEQF$ so that only when $ar(OLT^k_{i,j}) + ex(OLT^k_{i,j}) > Opt$, we derive $dl(OLT^k_{i,j})$ by EQF. Otherwise, we set $dl(OLT^k_{i,j}) = Opt$. We call this method, the **SLALL** method.

D. Scaling based on EQF

A problem common to both $SLEQF$ and $SLALL$ is that the height of a TUF is not changed, which may convey inaccurate information to the node-local scheduler with the sub-TUF. Thus, we derive methods to change the height of a TUF based on its deadline slicing. Similar to $SLEQF$, we can first derive the deadline of an OLT using EQF. Then, to obtain the TUF of the OLT, the height of the DT's TUF can be scaled by the factor: $\frac{dl(OLT_{i,j}^k)-ar(OLT_{i,j}^k)}{dl(DT_k)-ar(DT_k)}$. We call this method **Scale on EQF (SCEQF)**. Figure 10 illustrates the method with an example DT whose segment number m is 3.

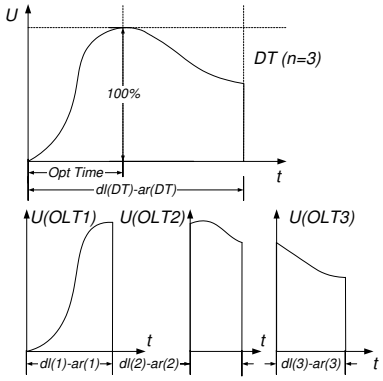


Fig. 9. The $SLEQF$ Technique

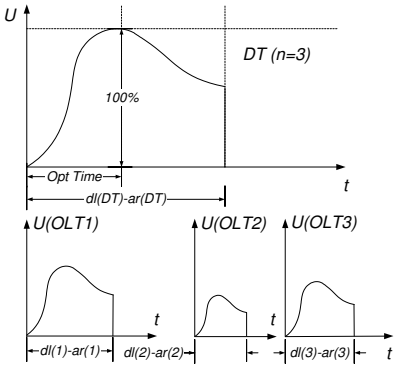


Fig. 10. The $SCEQF$ Technique

E. Scaling based on TUF shape

Some algorithms compute scheduling decisions that are mainly based on the potential value density (or PUD) of the threads. Thus, we can change the shape and height of the TUF assigned to an OLT such that the OLT's PUD is positively influenced. This will result in the OLT being favored by the scheduling algorithm over others.

Considering *Opt* in the TUF of a DT, and making the similar modification to $SCEQF$ as we have done in the process from $SLEQF$ to $SLALL$, we can derive the strategy of **SCALL** from $SCEQF$.

F. Decomposing into Linear-Constant TUF

We can also change the PUD of an OLT by decomposing the DT's TUF into a linear-constant TUF. Unlike $SLEQF$, $SLALL$, $SCEQF$ and $SCALL$, we don't slice the deadline of a DT to allocate to its OLTs in this strategy. This method, which is called **OptValue and Constant Value Function (OPTCON)**, is illustrated in Figure 11. The terms Opt and $OptValue$ have the same meanings as before. As shown in the figure, for DT_k and $OLT_{i,j}^k$, we first set $dl(OLT_{i,j}^k) = dl(DT_k)$. The times t_1 and t_2 are defined as $t_1 = ar(OLT_{i,j}^k)$ and $t_2 = ar(OLT_{i,j}^k) + ex(OLT_{i,j}^k)$.

$OPTCON$ decomposes the TUF of a DT to allocate sub-TUFs to OLTs with the following steps (the steps are also illustrated in Figure 11):

A Let the $OLT_{i,j}^k$ obtain its highest utility at its expected finish time t_2, which means $U(OLT_{i,j}^k, t_2) = OptValue$.

B We then increase the TUF of the OLT from time t_1 with utility value $U(OLT_{i,j}^k, t_1)$ linearly until the utility value $OptValue$ at time t_2, after which we keep the TUF constant at $OptValue$ until its deadline time $dl(OLT_{i,j}^k)$.

C The TUF of the $OLT_{i,j}^k$ is then scaled by a factor $fctr$ so that the expected PUD of the OLT is increased. The factor is determined as: $fctr = max\left(\frac{PUD_{local}}{PeakPUD_{OLT_{i,j}^k}}, 1\right)$, where $PeakPUD_{OLT_{i,j}^k} = \frac{OptValue}{ex(OLT_{i,j}^k)}$, and PUD_{local} is the PUD of local threads. Therefore, $fctr$ is chosen in such a way that an OLT's TUF is not scaled if its peak PUD is larger than the PUD of local threads, otherwise it is scaled *up* by the ratio of PUDs of local threads and the OLT. But the OLT's TUF will never be scaled down.

G. Decomposing into Linear TUF

To respect the shape of the propagated TUF in the process of decomposition, it is desirable to slightly modify the $OPTCON$ strategy as follows: For DT_k with m_k segments, let $OLT_{i,j}^k$ denote the ith OLT. Then, we can set $dl(OLT_{i,j}^k) = dl(DT_k)$, and still define $t_1 = ar(OLT_{i,j}^k)$ and $t_2 = ar(OLT_{i,j}^k) + ex(OLT_{i,j}^k)$.

We call this method **OptValue and Linear Function (OPTLNR)** and it is illustrated in Figure 12. The steps followed by the method include:

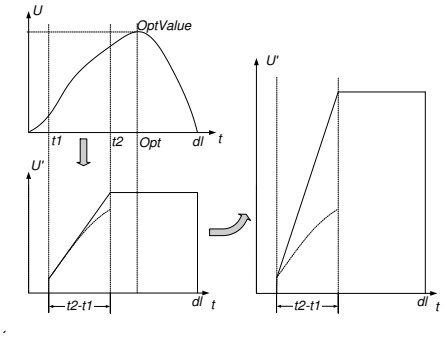


Fig. 11. The $OPTCON$ Technique

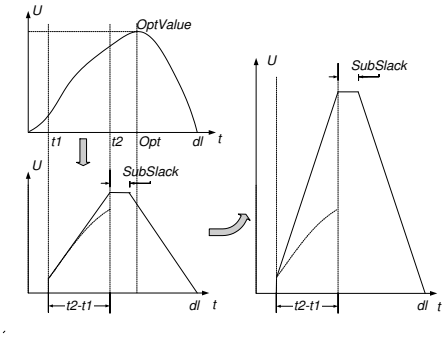


Fig. 12. The $OPTLNR$ Technique

A Let $OLT^k_{i,j}$ obtain its highest utility at its expected finish time t_2, which means $U(OLT^k_{i,j}, t_2) = OptValue$.

B We then increase the TUF of the OLT from time t_1 with utility value $U(OLT^k_{i,j}, t_1)$ linearly until the utility value $OptValue$ at time t_2, after which the TUF is kept constant at $OptValue$ for some period of time, denoted as $SubSlack$.

C After the $SubSlack$ period, we decrease the TUF from $OptValue$ linearly until it reaches zero utility at it deadline time $dl(OLT^k_{i,j})$.

D The $SubSlack$ is decided by comparing the expected finish time of the DT with the Opt time of DT_k. That is, we consider $slack = Opt - ar(OLT^k_{i,j}) - \sum_{l=i}^{m_k} ex(OLT^k_{l,j})$. If $slack$ is less than zero, we then set $SubSlack$ to be zero; otherwise, we compute $SubSlack$ using a method similar to that of EQF. The formula to calculate $SubSlack$ is described as follows:

$$SubSlack = \left(Opt - ar(OLT^k_{i,j}) - \sum_{l=i}^{m_k} ex(OLT^k_{l,j}) \right) \times \left(\frac{ex(OLT^k_{i,j})}{\sum_{l=i}^{m_k} ex(OLT^k_{l,j})} \right) \quad (1)$$

E Finally, we scale the whole TUF of the OLT by a factor $fctr$ to obtain the decomposed TUF of the OLT. The factor $fctr$ is defined as: $fctr = max\left(\frac{PUD_{local}}{PeakPUD_{OLT^k_{i,j}}}, 1\right)$, where $PeakPUD_{OLT^k_{i,j}} = \frac{OptValue}{ex(OLT^k_{i,j})}$. We calculate $fctr$ here in the same manner as $OPTCON$.

H. Scaling the TUF

In stead of the complicated decomposition methods described before, we also tried a simple decomposition strategy to improve the OLT's PUD that is seen by the local scheduler. This

is realized by: (1) using the DT's deadline as the OLT's deadline; and (2) scaling the DT's propagated TUF and assigning the scaled TUF as the OLT's TUF. The scaling of the DT's TUF can be done by using an $fctr$ factor that is determined exactly the same as in the $OPTCON$ and $OPTLNR$ techniques. We call this strategy **Time/Utility Function Scaling (TUFS)**.

I. Scaling into Rectangular TUF

Finally, we consider the extreme case of improving the OLT's PUD. The DT's deadline is kept the same as the OLT's deadline. However, the shape of the DT's TUF is changed into a step downward TUF and is scaled by a "large" factor $fctr$. In our experiment, this $fctr$ is selected such that the height of scaled TUF is larger than $OptValue$ of original TUF by an order of magnitude. The resulting TUF is then assigned to the OLT. Thus, irrespective of shape of the DT's TUF, the OLTs are assigned a step downward TUF with a large height. We call this method **Step Downward Function Scaling (STEPS)**.

In contrast to the strategies discussed previously, $STEPS$ assigns the same TUF to each OLT of a DT, and may radically change the global-global and global-local contention. Although it is not a systematic method for decomposing TUF's, it may still help us to understand the conditions under which TUF decomposition can improve DT performance.

VI. Experiment Evaluation

We experimentally study the TUF decomposition strategies by conducting simulation experiments. We believe that simulation is an appropriate tool for this study as that would allow us to evaluate a number of different decomposition techniques under a broad range of system conditions.

We first present the simulation model, and then discuss the experimental results.

A. Simulation Model

Our simulator is written with the simulation tool OMNET++ [21], which provides a discrete event simulation environment. Each simulation experiment (generating a single

data point) consists of three simulation runs with different random seeds, each lasting 200 sec (at least 10,000 events are generated per run; many more for high load experiments). Since the basic time unit in OMNET++ is a second, we will refer to a time unit as a second hereafter. The structure of our simulation model follows the conceptual model described in Section III, with the following characteristics:

- **Nodes.** There are k homogeneous nodes in the system. Each node services their threads (OLTs of DTs and local threads) according to a given real-time scheduling algorithm. We consider both UA and non-UA scheduling algorithms such as GUS, DASA, LBESA, D^{over}, EDF and LLF for our experiments.
- **Local threads.** Local threads are generated at each node according to a Poisson distribution with mean inter-arrival time $1/\lambda_{local}$ seconds. (Poisson distributions are typically used in analytical studies like ours because of their simplicity and because they yield useful insights.) Since there are k nodes, the total average arrival rate is k/λ_{local} per second. Execution times of local threads are exponentially distributed with mean in $1/\mu_{local}$ seconds. The rate of work due to local threads is thus $k\lambda_{local}/\mu_{local}$.
- **DTs.** Similar to local threads, DTs are generated as n streams of Poisson processes with mean inter-arrival $1/\lambda_{DT}$ time. For simplicity, we assume that DTs are homogeneous. In particular, we assume that all DTs consist of the same number of segments, and the execution times of all the segments (OLTs) follow the same exponential distribution with a mean $1/\mu_{OLT}$ seconds. We assume that the number of OLTs contained in a DT is m, i.e., $m_k = m, \forall k \in \{1, 2, ..., n\}$. The total execution times of DTs thus follow an m-stage Erlang distribution with mean m/μ_{OLT}. The rate of work due to DTs is therefore $m\lambda_{DT}/\mu_{OLT}$. The execution node of an OLT is selected randomly and uniformly from the k nodes.
- **System Load.** We define the *normalized load* (or *load*, for short) as the ratio of the rate of work generated to the total processing capacity of the system. That is,

$$load = \left(\frac{n \cdot m \cdot \lambda_{DT}}{\mu_{OLT}} + \frac{k \cdot \lambda_{local}}{\mu_{local}} \right) \Big/ k \tag{2}$$

For a stable system, we have $0 \leq load \leq 1$. We also define $frac_local$ as the least fraction

of PUD that can possibly be contributed by local threads in node-local scheduling, i.e.,

$$frac_local = \frac{PUD_{local}}{PUD_{local} + PeakPUD_{OLT}} \tag{3}$$

• **TUFs.** Different threads have different TUFs. Each DT has a propagated TUF, and we define five classes of TUFs to evaluate our methods. The parameter setting of our baseline experiment is summarized in Table I. In Table I, TUF_1 and TUF_2 are non-increasing, TUF_3 is a step downward function, TUF_4 is strictly concave, and TUF_5 is the combination of different TUFs. In particular, TUF_1 is the right half of a quadratic function in the first quadrant, TUF_2 is a linear function, and TUF_4 is a complete quadratic function in the first quadrant. The TUFs of local threads are step downward functions with variable heights.

TABLE I

Baseline Settings

Overload Mgmt. Policy	Abort threads later than deadlines
Local Scheduler	GUS, DASA, LBESA, D^{over}, EDF, LLF
μ_{OLT}	1.0 (When *load* varies)
μ_{local}	1.0 (When *load* varies)
λ_{DT}	1/8.33 (When GSF varies)
λ_{local}	1/8.33 (When GSF varies)
k (#of nodes)	8
m (# of OLTs in a DT)	4
n (# of DT streams)	3
$frac_local$	0.80 (When *load* or GSF varies)
TUF_1	$f_1(t) = \begin{cases} -0.025t^2 + 10, & \text{when } 0 \le t \le 20 \\ 0, & \text{otherwise} \end{cases}$
TUF_2	$f_2(t) = \begin{cases} -0.5t + 10, & \text{when } 0 \le t \le 20 \\ 0, & \text{otherwise} \end{cases}$
TUF_3	$f_3(t) = \begin{cases} 10, & \text{when } 0 \le t \le 20 \\ 0, & \text{otherwise} \end{cases}$
TUF_4	$f_4(t) = \begin{cases} -0.1t^2 + 2t, & \text{when } 0 \le t \le 20 \\ 0, & \text{otherwise} \end{cases}$
TUF_5	combination of $TUF_1 \sim TUF_4$
Local threads	Step Downward Functions

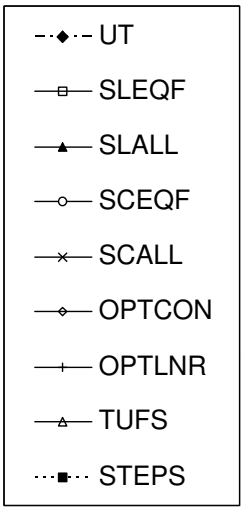


Fig. 13. Legend

B. Experimental Results

The primary performance metric that we use to evaluate the TUF decomposition methods is utility accrual ratio (or UR), which is defined as the ratio of the accrued utility to the maximum possible total utility. The maximum possible total utility is the sum of each DT's maximum utility.

We use the notation $UR_A^B(C)$ to denote the utility ratio obtained under a scheduling algorithm $A \in \{GUS, DASA, LBESA, D^{over}, EDF, LLF\}$, a decomposition technique $B \in \{UT, SLEQF, SLALL, SCEQF, SCALL, OPTCON, OPTLNR, TUFS, STEPS\}$, and a TUF $C \in \{TUF_1, TUF_2, TUF_3, TUF_4, TUF_5\}$. Thus, for example, $UR_{GUS}^{SLEQF}(TUF_4)$ denotes the utility ratio that DTs can accrue under the GUS scheduler, $SLEQF$ decomposition technique, and TUF_4.

In Section IV, we evaluate the effects of all factors presented on system performance in a collective way, so all simulations are performed with variation of several parameters describing the factors. The legends for curves derived from simulation with nine different decomposition strategies are shown in Figure 13.

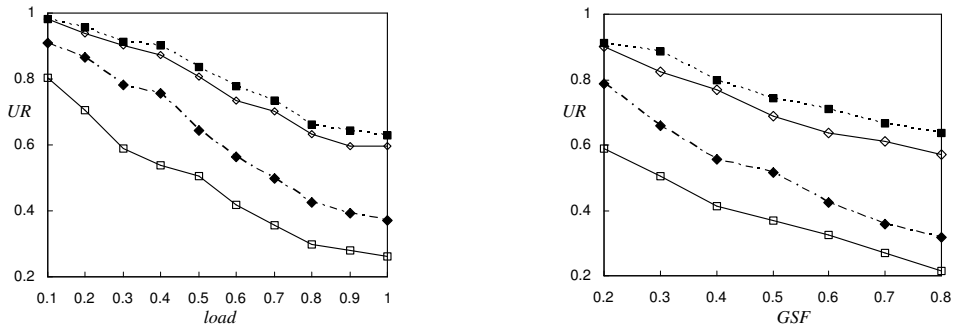
1) Effect of Schedulers on Performance: We first study the impact of scheduling algorithms on TUF decomposition methods. We consider step downward functions (i.e., TUF_3) for the DTs, as all the algorithms, with the exception of GUS and LBESA, cannot deal with arbitrarily-shaped TUFs. Further, we do not consider resource dependencies, as LBESA, D^{over}, EDF and LLF cannot directly address resource dependencies.

The TUFs of local threads are set to step downward functions with heights of 40, and the maximum heights of DT TUFs are bounded at 10. Thus, the $frac_local$ in these experiments is 0.8. Since we only focus on TUF_3 here, for simplicity, we denote $UR_A^B(TUF_3)$ as UR_A^B. URs of different methods are recorded as the *load* and *GSF* varies from 0 to 1. As shown in Table I, when *load* varies, we keep the execution time $\mu_{OLT} = \mu_{local} = 1.0$, but change the mean inter-arrival time $1/\lambda_{DT}$ and $1/\lambda_{local}$; when GSF varies, we keep $\lambda_{DT} = \lambda_{local} = 1/8.33$, but change μ_{OLT} and μ_{local}.

Figures 14(a) and 14(b) show UR_{GUS} of various decomposition strategies as *load* and *GSF* varies. In Figure 14, we do not show the nine curves of decomposition strategies because similar results are combined.

UR_{GUS}^{UT}, UR_{GUS}^{SLALL} and UR_{GUS}^{SCALL} are identical because the three strategies perform the same operations with step downward functions, so we only show UR_{GUS}^{UT} to represent them. For the same reason UR_{GUS}^{SLEQF} and UR_{GUS}^{SCEQF} are identical so we keep the former. In addition, UR_{GUS}^{OPTCON}, UR_{GUS}^{OPTLNR} and UR_{GUS}^{TUFS} are identical, so we only keep UR_{GUS}^{OPTCON}.

Hereafter, we will use "≈" to describe very similar curves, and use "=" to describe identical curves. Thus, if $Line_1 \approx Line_2$ or $Line_1 = Line_2$, then we only show $Line_1$ in the figure to represent both of them. This is the same with multiple similar or identical curves.



(a) UR vs. $load$ (b) UR vs. GSF

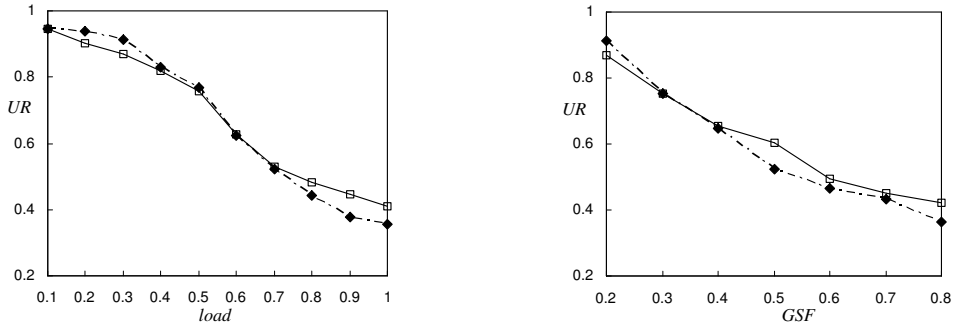
Fig. 14. Utility Ratio with GUS under TUF_3

From Figures 14(a) and 14(b), we observe that the curves in both figures indicate similar trends. For example, in Figure 14(a), when *load* increases, UR_{GUS} of all decomposition strategies decreases. This is reasonable, because more threads miss their deadlines and thus less utilities are accrued. But GUS under different decomposition strategies accrues different utilities, even when the *load* is very light. UR_{GUS}^{SLEQF} and UR_{GUS}^{SCEQF} have the worst performance among all strategies. UR_{GUS}^{SLALL} and UR_{GUS}^{SCALL} are better than the former two, but perform as well as UR_{GUS}^{UT}. The methods $OPTCON$, $OPTLNR$ and $TUFS$ perform better than UT, but worse than $STEPS$, which is the best among all methods.

For example, at $load = 0.6$, UR_{GUS}^{SCEQF} and $UR_{GUS}^{SLEQF} \approx 42\%$, UR_{GUS}^{SCALL}, UR_{GUS}^{SLALL} and $UR_{GUS}^{UT} \approx 56\%$, UR_{GUS}^{OPTCON}, UR_{GUS}^{OPTLNR} and $UR_{GUS}^{Ulti_Sclg} \approx 73\%$, and UR_{GUS}^{STEPS} is 78%.

The performance results of the decomposition strategies under the LBESA algorithm are shown in Figures 15(a) and 15(b). In both figures, $UR_{LBESA}^{SLEQF} = UR_{LBESA}^{SCEQF}$; so we only show UR_{LBESA}^{SLEQF}. Furthermore, the other seven strategies produce almost identical results; so we only show UR_{LBESA}^{UT}. With LBESA, the $SLEQF$ and $SLALL$ strategies perform better than others, especially at high *load* and GSF.

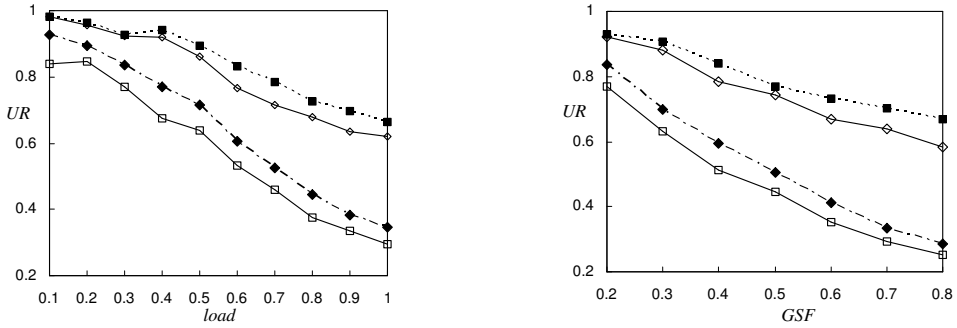
Figures 16(a) and 16(b) show the corresponding performance under the DASA algorithm. In Figure 16, we note that $UR_{DASA}^{SLEQF} = UR_{DASA}^{SCEQF}$, $UR_{DASA}^{UT} = UR_{DASA}^{SLALL} = UR_{DASA}^{SCALL}$, and



(a) UR vs. $load$ (b) UR vs. GSF

Fig. 15. Utility Ratio with LBESA under TUF_3

$UR^{OPTCON}_{DASA} = UR^{OPTLNR}_{DASA} = UR^{TUFS}_{DASA}$. Observe that under DASA, as $load$ and GSF varies, $STEPS$ always performs better than all the others. $OPTCON$, $OPTLNR$, and $TUFS$ perform better than UT and $SLEQF$. Furthermore, $SCEQF$ shows the worst performance.

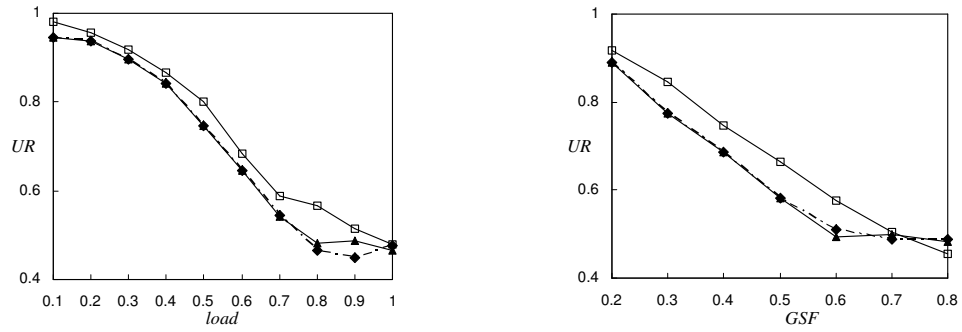


(a) UR vs. $load$ (b) UR vs. GSF

Fig. 16. Utility Ratio with DASA

The performance of the LLF and EDF algorithms under the different decomposition strategies are shown in Figures 17 and 18. The results of LLF and EDF show similar trends. Under both LLF and EDF, we observe that the results of $SLEQF$ and $SCEQF$ are identical to each other. Thus, we only show $UR^{SLEQF}_{LLF/EDF}$. The results of $SLALL$ and $SCALL$ are the same. So we show $UR^{SLALL}_{LLF/EDF}$. All the other decomposition methods have exactly the same performance and therefore are represented by $UR^{UT}_{LLF/EDF}$.

Under the LLF algorithm, $SLEQF$ and $SCEQF$ perform better than the others, except that at very high $load$ and GSF, the curves converge. Under EDF, $SLEQF$ and $SCEQF$



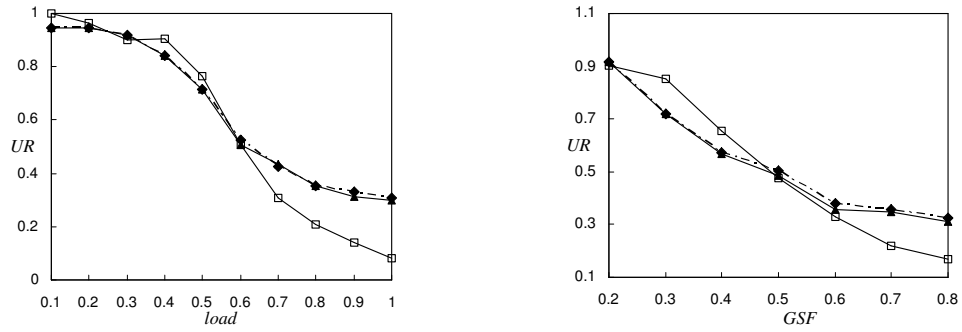
(a) UR vs. $load$ (b) UR vs. GSF

Fig. 17. Utility Ratio with LLF

perform better than the others when $load$ or GSF is less than 0.5.

LLF and EDF are not UA schedulers, but the primary performance measure used by them, i.e., deadline miss ratio, can be converted to utility ratio. Intuitively, the more deadlines are met by DTs, the more utilities are accrued. Thus, the deadline miss ratio performance metric of LLF and EDF can be a reasonable metric for UA scheduling.

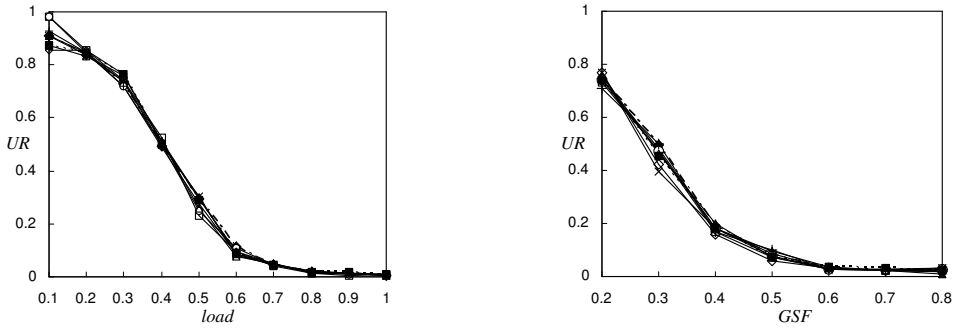
The results under EDF shown in Figure 18 is consistent with Kao's experimental results presented in [20], where the deadline miss ratio of EQF is lower than the ultimate deadline (UD) strategy as $load$ varies from 0.1 to 0.5.



(a) UR vs. $load$ (b) UR vs. GSF

Fig. 18. Utility Ratio with EDF

Figures 19 shows the performance of decomposition strategies under the D^{over} algorithm. We observe that all the curves are very close to each other. This indicates that the different decomposition strategies have little effect on D^{over}'s scheduling decisions.



(a) UR vs. $load$ (b) UR vs. GSF

Fig. 19. Utility Ratio with D^{over}

Different scheduling algorithms compute scheduling decisions using different metrics. Thus, they differ in their resulting behaviors. For example, EDF exclusively considers a thread's deadline, and suffers significant domino effects during overloads. DASA, on the other hand, considers both deadlines and PUDs, and exhibits good performance during overload situations. Thus, the performance of a given decomposition method will be differently influenced by different underlying scheduling algorithms. For this reason, large performance gaps, in terms of URs, are particularly interesting to us.

To understand such differences, we consider three types of resource competition among threads. These include local-local, local-global, and global-global. A scheduling algorithm resolves the contention mainly by its scheduling metric. Scheduling metrics include deadline (for EDF), laxity (for LLF), PUD (for GUS and DASA), a timer value (for D^{over}) and combination of some metrics (for LBESA and DASA). Thus, we can loosely categorize schedulers into deadline-based, laxity-based, PUD-based and timer-based.

The TUF decomposition methods that we consider here can also be loosely categorized into three classes: (1) those that change no properties (UT), (2) those that change an OLT's deadline time ($SLEQF$, $SLALL$, $SCEQF$ and $SCALL$), and (3) those that change an OLT's PUD ($OPTCON$, $OPTLNR$, $TUFS$, $STEPS$). Different TUF decomposition strategies alter the OLTs' metrics used by the schedulers, so they only impact local-global and global-global contention. DTs are subject to local-global and global-global contention. Thus, the choice of a decomposition strategy significantly affects them, affecting the UR.

GUS resolves resource contention mainly by comparing the PUDs of threads. Our simulation reveals that, under GUS, the performance of those decomposition strategies that increase OLTs' PUDs are better than that of others, for various parameter settings.

As an extreme case, the unsystematic strategy, $STEPS$, always performs the best because it seeks to increase each OLT's PUD so that it is much larger than those of local threads. We may notice that slicing the deadline of a DT and allocating sub deadlines to its OLTs can also increase the OLTs' PUDs. But even with such deadline slicing techniques, DTs still perform poorly. The reason is that a DT consists of a series of OLTs, and if any OLT in the DT misses its sub deadline and is aborted by the scheduler, the parent DT fails. Thus, deadline slicing techniques sometimes can be detrimental to DTs, if we too "tight" sub deadline constraints are assigned to their OLTs.

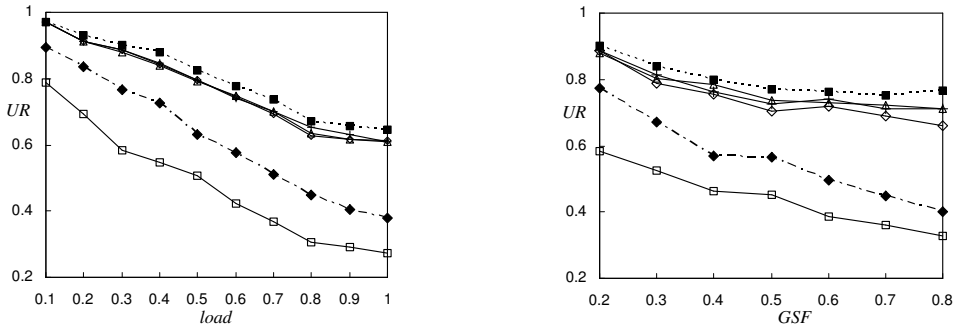
It is interesting to observe that D^{over}, as a timer based scheduling algorithm, is almost not affected by TUF decomposition. The decomposition strategies with deadline slicing perform well under EDF and LLF, because deadline slicing gives OLTs shorter deadline times, which in turn gives them higher priorities for accessing resources under EDF and LLF. Both LBESA and DASA compare deadlines and PUDs for scheduling, but in different ways. From their performance under different TUF decomposition methods, we can infer that DASA is more "PUD-based" and LBESA is more "deadline-based."

2) Effect of TUF shape on Performance: Different shapes of TUFs can be decomposed differently. This can potentially yield different performance under different decomposition strategies. Thus, in this section, we study the effect of decomposing TUFs with different shapes. Our study focuses on the GUS and LBESA algorithms, since they can deal with arbitrarily-shaped TUFs.

Besides step downward TUF (TUF_3), we also conducted experiments with non-increasing TUFs (TUF_1 and TUF_2), and strictly concave TUF (TUF_4). To obtain an average performance of the decomposition methods on various TUFs, we also considered TUF_5, which is a combination of different TUFs.

Figures 20 and 32 show the URs under the GUS algorithm for TUF_1 and TUF_2, respectively. We show some of the results in Appendix A. From Figure 20, we observe that

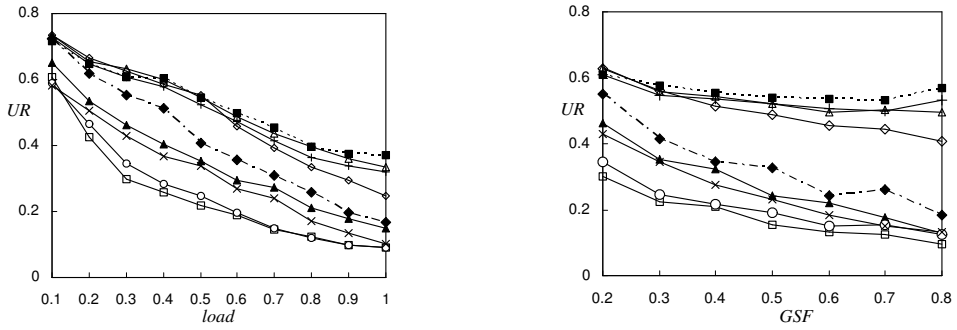
when *load* and GSF varies, $UR_{GUS}^{SLEQF}(TUF_1) = UR_{GUS}^{SLALL}(TUF_1) \approx UR_{GUS}^{SCEQF}(TUF_1) = UR_{GUS}^{SCALL}(TUF_1)$.



(a) UR vs. *load* (b) UR vs. GSF

Fig. 20. Utility Ratio with GUS under TUF_1

Figure 21 shows the results of GUS under TUF_4. In contrast to the results under non-increasing and step downward TUFs, different TUF decomposition strategies show different performance, respectively, with strictly concave TUFs. But at any *load* or GSF, $STEPS$ performs the best, while $SLEQF$, $SLALL$, $SCEQF$ and $SCALL$ performs worse than others. The average performance of the decomposition methods under the combination of different TUFs is shown in Figure 33 of Appendix A.



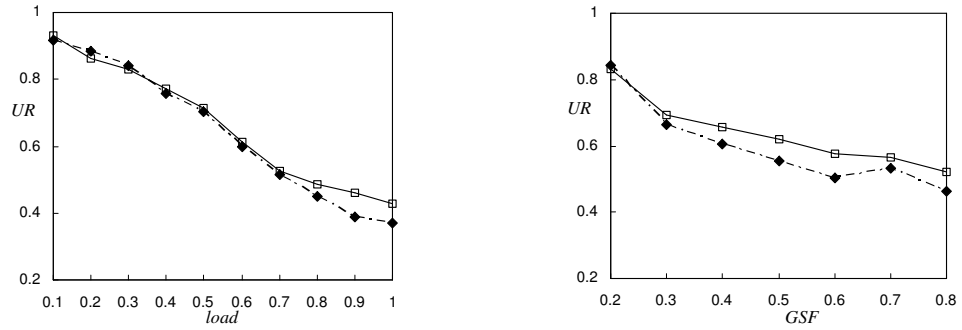
(a) UR vs. *load* (b) UR vs. GSF

Fig. 21. Utility Ratio with GUS under TUF_4

We show the corresponding results of the LBESA with TUF_1 and TUF_4 in Figures 22 and 23, respectively. In Appendix A, we show the results with TUF_2 and TUF_5.

We observe in Figure 22 that, $UR_{LBESA}^{SLEQF}(TUF_1) \approx UR_{LBESA}^{SLALL}(TUF_1) \approx UR_{LBESA}^{SCEQF}(TUF_1)$

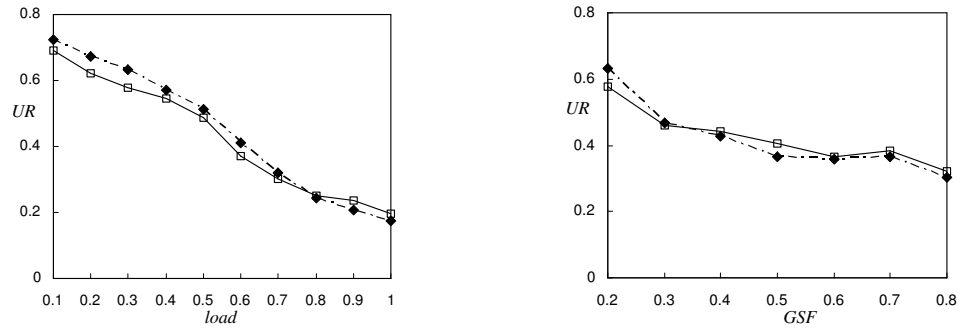
$\approx UR^{SCALL}_{LBESA}(TUF_1)$. All other results are close to each other. With LBESA and increasing TUFs, the four strategies of $SLEQF/ALL$ and $SCEQF/ALL$ perform better than others, especially at higher *load* and GSF.



(a) UR vs. *load* (b) UR vs. GSF

Fig. 22. Utility Ratio with LBESA under TUF_1

However, with strictly concave TUFs, we obtain different results. As shown in Figure 23, where $UR^{SLEQF}_{GUS}(TUF_4) \approx UR^{SCEQF}_{GUS}(TUF_4)$, and the other results are similar, the strategies of $SLEQF$ and $SCEQF$ are performing no better than others. Even when we vary *load* and GSF, their performance is worse than the others.



(a) UR vs. *load* (b) UR vs. GSF

Fig. 23. Utility Ratio with LBESA under TUF_4

Among the three shapes of TUFs considered in our experiments, strictly concave TUFs have the most apparent impact on TUF decomposition. From our experiments, we observe that, with GUS and TUF_4, the performance of $OPTCON$ and $OPTLNR$ is better than others, except with the unsystematic method $STEPS$. Further, the performance of

$OPTCON$ is very close to that of $OPTLNR$ over a wide range of parameter settings. But in case when they differ, $OPTLNR$ is usually superior. With large *load* and GSF, $OPTLNR$ is performing better than $OPTCON$. This is not surprising because $OPTLNR$ more accurately reflects the shape of the original TUF of a DT than $OPTCON$. Thus, the technique improves the OLT's chances for accessing the resources as well as conveys more accurate information to GUS.

Although the LBESA algorithm uses the PUD metric, it is not the deciding metric in its scheduling decisions. The algorithm's deadline-ordering of threads also impacts the final scheduling order. Thus, for $OPTCON$ and $OPTLNR$, which seek to improve OLTs' PUDs while respecting the TUF shapes, their effects cannot be observed with LBESA and TUF_4.

3) Effect of Local Threads on Performance: Comparing the URs of the different decomposition strategies, we infer that as the *load* and GSF increases, the performance of all strategies deteriorate. However, all of our previously reported experiments are carried out with $frac_local = 0.80$.

We had hypothesized that by changing the deadlines or PUDs of OLTs, various TUF decomposition strategies can help DTs to "grab" resources (including CPU) from local threads, since deadlines or PUDs are key scheduling metrics used by the scheduling algorithms. For example, the likelihood of DTs to obtain resources can be improved by reducing the deadlines of OLTs (under EDF and LLF algorithms) or by increasing the PUDs of OLTs (under GUS and DASA algorithms).

To verify this hypothesis, in this section, we vary the relative proportion of the two kinds of threads, i.e., we vary $frac_local$ from 0.3 to 0.88 and study the $UR(TUF_3)$ of the different algorithms. In these experiments, we keep $\mu_{OLT} = \mu_{local} = 1.0$ and $\lambda_{DT} = \lambda_{local} = 1/8.33$. For simplicity in notation, we still use UR_A^B to represent $UR_A^B(TUF_3)$.

Figures 24 and 25 show the performance of GUS and DASA with different TUF decomposition strategies. We observe that GUS and DASA produce similar performance. In both figures, $UR_{GUS/DASA}^{UT} = UR_{GUS/DASA}^{SLALL} = UR_{GUS/DASA}^{SCALL}$, $UR_{GUS/DASA}^{SLEQF} = UR_{GUS/DASA}^{SCEQF}$, and $UR_{GUS/DASA}^{OPTCON} = UR_{GUS/DASA}^{OPTLNR} = UR_{GUS/DASA}^{TUFS}$.

From Figures 24 and 25, we observe that as $frac_local$ increases (i.e., more PUD is

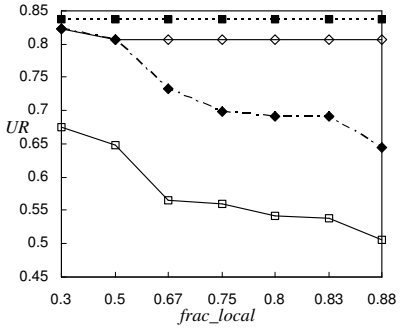


Fig. 24. UR vs. $frac_local$ with GUS under TUF_3

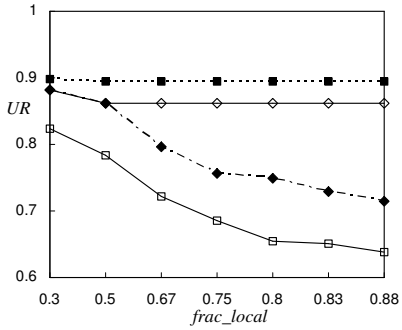


Fig. 25. UR vs. $frac_local$ with DASA under TUF_3

contributed by local threads), $UR^{UT}_{GUS/DASA}$ drops dramatically. When $frac_local$ is less than or equal to 0.5, and when GUS and DASA are used for scheduling OLTs and local threads, the two classes of threads have almost equal PUDs and thus almost equal chances to be selected to execute.

Intuitively, UT should perform similarly as other decomposition techniques that increase PUDs of DTs, but better than deadline slicing techniques. This is correctly reflected in the plots. But when $frac_local$ is large than 0.5, OLTs with sub-TUFs allocated by UT will be at a disadvantage in the PUD-based scheduling process of GUS and DASA. Thus, we observe that higher the $frac_local$, the worse UT performs. Furthermore, the performance of $OPTCON$, $OPTLNR$, and $TUFS$ are almost a constant as $frac_local$ increases, because these decomposition methods always keep PUDs of OLTs comparable with PUDs of local threads. Note that $STEPS$ always performs the best independent of $frac_local$, because it "unfairly" increases PUDs of OLTs.

Figures 26 and 27 show the performance of LBESA and D^{over} algorithms under different decomposition strategies. In Figure 26, $UR^{UT}_{LBESA} = UR^{SLALL}_{LBESA} = UR^{SCALL}_{LBESA}$, $UR^{SLEQF}_{LBESA} = UR^{SCEQF}_{LBESA}$, and $UR^{OPTCON}_{LBESA} = UR^{OPTLNR}_{LBESA} = UR^{TUFS}_{LBESA} = UR^{STEPS}_{LBESA}$.

In Figure 27, we observe that all curves converge in a narrow zone between 0.24 and 0.29. In Figure 26, we observe that $OPTCON$, $OPTLNR$ and $TUFS$ are almost constant as $frac_local$ increases, but only slightly outperform UT. This is reasonable, because the

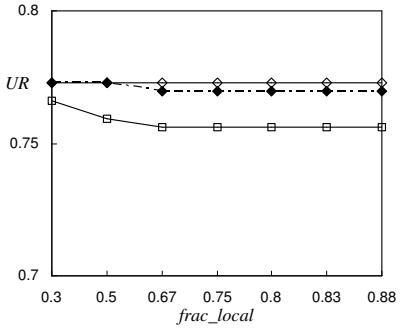


Fig. 26. UR vs. $frac_local$ with LBESA under TUF_3

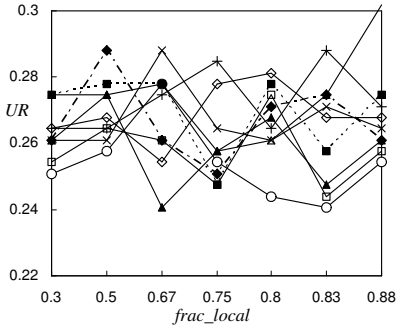


Fig. 27. UR vs. $frac_local$ with D^{over} under TUF_3

scheduling decisions made by LBESA are only partially dependent on PUDs. On the other hand, D^{over} does not consider PUDs. Thus, the results of D^{over} do not exhibit any regular pattern in Figure 27.

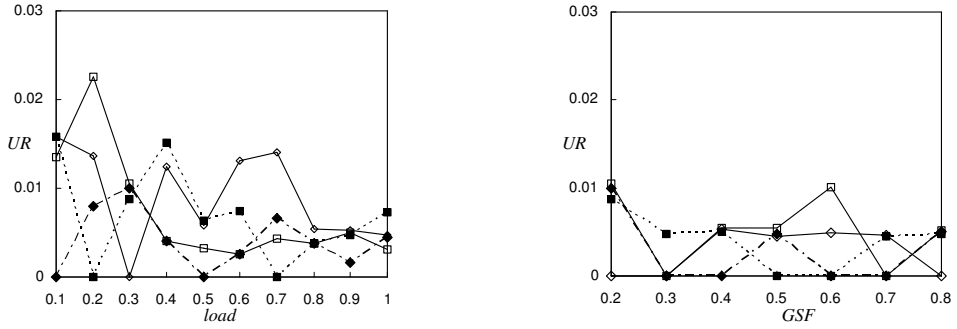
The performance of various decomposition strategies under the LLF and EDF algorithms is not affected by $frac_local$. This is because, varying $frac_local$ only changes the PUD contributed by the local threads. This does not affect the scheduling decisions made by LLF and EDF.

4) Effect of Dependencies on Performance: There are no resource dependencies among OLTs in the experiments that we have conducted so far. In this section, we study how resource dependencies among OLTs affect the performance of TUF decomposition strategies.

We impose resource dependencies among OLTs of different DTs. We then study the performance of the decomposition strategies under the GUS and DASA algorithms, as only these two algorithms can deal with resource dependencies. For OLTs within a single DT, there are no resource dependencies, but only precedence dependencies.

We first consider the step downward function, TUF_3, for these experiments. Subsequently, we study other TUF shapes.

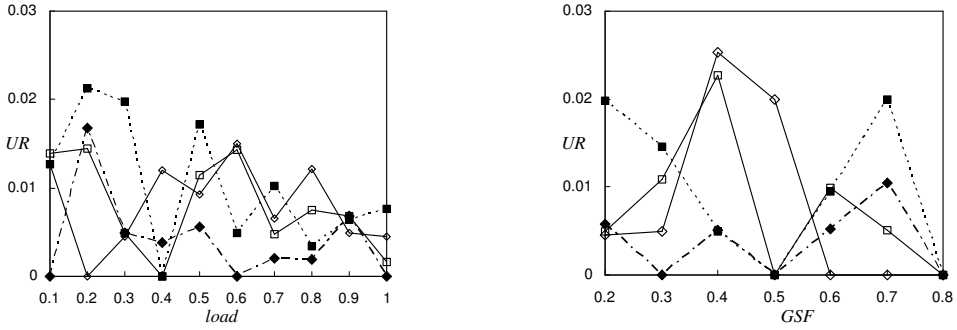
Figures 28 and 29 show UR_{GUS} and UR_{DASA}, respectively. GUS and DASA also bear similar trends with resource dependencies among OLTs. From both Figures 28 and 29, we can infer that $UR^{UT}_{GUS/DASA} = UR^{SLALL}_{GUS/DASA} = UR^{SCALL}_{GUS/DASA}$, $UR^{SLEQF}_{GUS/DASA} = UR^{SCEQF}_{GUS/DASA}$,



(a) UR vs. $load$ (b) UR vs. GSF

Fig. 28. Utility Ratio with GUS and Resource Dependencies under TUF_3

and $UR^{OPTCON}_{GUS/DASA} = UR^{OPTLNR}_{GUS/DASA} = UR^{TUFS}_{GUS/DASA}$.



(a) UR vs. $load$ (b) UR vs. GSF

Fig. 29. Utility Ratio with DASA and Resource Dependencies

We also vary $frac_local$ to study the impact of resource dependencies on performance. Figures 30 and 31 show the URs of GUS and DASA with dependencies, as $frac_local$ varies, respectively. From both figures, we infer that $UR^{SLEQF}_{GUS/DASA} = UR^{SCEQF}_{GUS/DASA}$, $UR^{UT}_{GUS/DASA} = UR^{SLALL}_{GUS/DASA} = UR^{SCALL}_{GUS/DASA}$, and $UR^{OPTCON}_{GUS/DASA} = UR^{OPTLNR}_{GUS/DASA} = UR^{TUFS}_{GUS/DASA}$.

From the figures, we observe that performance drops when there are resource dependencies; URs of GUS and DASA under different decomposition strategies all decrease to less than 1%. Furthermore, we cannot see any regular pattern in the results when varying *load*, GSF, and $frac_local$. This is reasonable because TUF decomposition for OLTs in a DT cannot accommodate significant interference from other OLTs. But resource dependencies and resource access operations among OLTs can cause unexpected, sometimes extremely

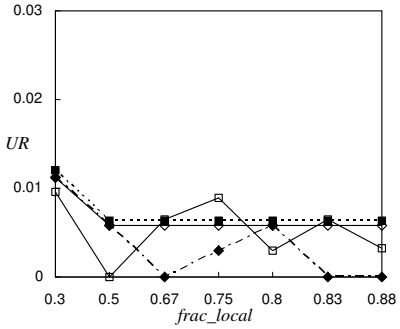


Fig. 30. UR vs. $frac_local$ with GUS and Resource Dependencies under TUF_3

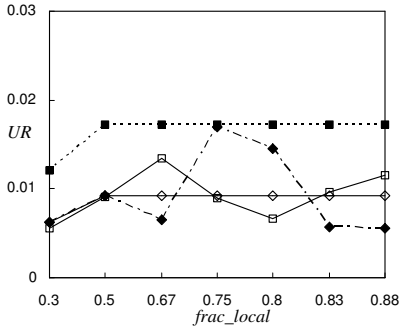


Fig. 31. UR vs. $frac_local$ with DASA and Resource Dependencies

large, interference to the decomposition strategies, which results in their poor performance.

We observed similar results for GUS with other TUF shapes, so they are not listed here. These plots are shown in Appendix B.

C. Summary of Experiments and Conclusions

In summary, our experiments analyze the possible factors that can affect the performance of TUF decomposition strategies in terms of utility accrued by DTs. We summarize our results and make conclusions with tables shown in this section.

In this section, we loosely categorize the decomposition strategies into four classes. UT, which changes no properties of a DT's TUF at all, is the baseline method. The strategies of $SLEQF$, $SLALL$, $SCEQF$ and $SCALL$ decide the deadlines of a DT's OLTs by slicing the DT's deadline in the decomposition process, so they are categorized in the class of *Deadline Slicing*. Instead, $OPTCON$, $OPTLNR$ and $TUFS$ don't slice a DT's deadline but scale up its TUF and allocate it to the OLTs. Thus, they are classified as *Shape Scaling* strategies. Finally, the decomposition results of a DT's TUF by $STEPS$ are step downward sub-TUFs whose heights are larger than $OptValue$ of the original TUF by an order of magnitude. We refer to it as the *Extreme* method.

The performance comparison (in terms of UR) of TUF decomposition strategies on step downward TUFs with different node-local schedulers is summarized in Table II. As shown in the table, URs of all decomposition methods drop when *load* or GSF increases. But

for different schedulers, the methods show various performance, in terms of how URs are dropping, compared to the baseline method, UT. Such difference is described by the cell contents of Table II, which show "same", "better", "worse", etc.. For example, *Shape Scaling* strategies perform better for GUS and DASA than UT, but work as well as UT for other schedulers; *Deadline Slicing* strategies perform better for deadline-based schedulers such as LLF and EDF, but are not suitable for PUD-based ones such as GUS and DASA.

TABLE II

TUF Decomposition Summary I—Increasing *load* and *GSF* with Step Downward TUFs

$frac_local$ = 0.8	Baseline	Deadline Slicing				Shape Scaling			Extreme
	UT	SLEQF	SLALL	SCEQF	SCALL	OPTCON	OPTLNR	TUFS	STEPS
GUS	drops	worse	same	worse	same	better			best
DASA	drops	worse	same	worse	same	better			best
LBESA	drops	better		same		same			same
LLF	drops	better	similar	better	similar	same			same
EDF	drops	better when $load \leq 0.5$	similar	better when $load \leq 0.5$	similar	same			same
D^{over}	drops	All strategies show little difference							

Table III shows the URs of different decomposition methods on various shapes of TUFs under GUS and LBESA. With all shapes of TUFs, *Shape Scaling* strategies have better performance under GUS, but can provide no improvement under LBESA. *Deadline Slicing* strategies perform worse with all shapes of TUFs under GUS and only TUF_4 (strictly concave TUF) under LBESA, but perform better with the other shapes under LBESA.

TABLE III

TUF Decomposition Summary II—Changing TUF shapes While Increasing *load*

	$frac_local$ = 0.8	Baseline	Deadline Slicing				Shape Scaling			Extreme
		UT	SLEQF	SLALL	SCEQF	SCALL	OPTCON	OPTLNR	TUFS	STEPS
GUS	TUF_1	drops	worse				better			best
GUS	TUF_2	drops	worse				better			best
GUS	TUF_3	drops	worse	same	worse	same	better			best
GUS	TUF_4	drops	worse				better	much better		best
GUS	TUF_4	drops	worse				better	much better		best
LBESA	TUF_1	drops	better				same			same
LBESA	TUF_2	drops	better				same			same
LBESA	TUF_3	drops	better		same		same			same
LBESA	TUF_4	drops	worse	same	worse	same	same			same
LBESA	TUF_5	drops	similar	better	similar	better	same			same

The effects to decomposition of step downward TUFs of increasing $frac_local$ are shown in Table IV. LLF, EDF and D^{over} are not affected by $frac_local$. But under GUS, DASA

and LBESA, the increase of $frac_local$ causes quick drops on URs of the baseline method UT. *Shape Scaling* strategies perform better while *Deadline Slicing* strategies perform worse than UT, because the former methods can improve the OLTs' PUDs seen by the node-local scheduler when $frac_local$ increases.

TABLE IV
TUF Decomposition Summary III—Increasing $frac_local$ with Step Downward TUFs

$load$ = 0.3	Baseline	Deadline Slicing				Shape Scaling			Extreme
	UT	SLEQF	SLALL	SCEQF	SCALL	OPTCON	OPTLNR	TUFS	STEPS
GUS	drops quickly	worse	same	worse	same	better (not affected by $frac_local$)			best
DASA	drops quickly	worse	same	worse	same	better (not affected by $frac_local$)			best
LBESA	not affected	worse	same	worse	same	slightly better			
LLF	Results are not affected by $frac_local$.								
EDF	Results are not affected by $frac_local$.								
D^{over}	The results exhibit no regular patterns.								

With resource dependencies among OLTs of different DTs, all TUF decomposition strategies show very poor performance because of the unexpected and large interference caused by and resource access operations. Thus, these results are not shown with tables.

VII. Past Work

There are relatively few studies on the TUF decomposition problem. Most of the past efforts on time constraint decomposition in real-time distributed systems focus on the deadline constraint. We summarize these efforts and contrast them with our work.

Bettati and Liu [22], [23] present an approach for scheduling pre-allocated flow-shop (sequential) tasks in a hard real-time distributed environment. In their model, global tasks consist of (same) set of subtasks to be executed on nodes in the same order. The goal is to devise efficient off-line algorithms for computing a schedule for the subtasks such that all deadlines are met (if such a schedule exists). In their work, local deadlines are assigned by distributing end-to-end deadlines evenly over tasks, and then tasks are non-preemptively scheduled using a deadline-based priority scheme.

Kao and Garcia-Molina present multiple strategies for automatically translating the end-to-end deadline into deadlines for individual subtasks in distributed soft real-time systems [20], [24]. They reduce the subtask deadline assignment problem (SDA) into two

subproblems: the serial subtask problem (SSP) and parallel subtasks problem (PSP). The authors present decomposition strategies called Ultimate Deadline (UD), Effective Deadline (ED), Equal Slack (EQS), and Equal Flexibility (EQF) for the SSP problem. Furthermore, they propose a strategy called DIV-x for the PSP problem. The techniques are aimed at systems with complete *a priori* knowledge of task-processor assignment.

Di Natale and Stankovic [25] presents the end-to-end deadline slicing technique for assigning slices to tasks using the critical-path concept. The strategy used for finding slices is to determine a critical path in the task graph that minimizes the overall laxity of the path. The slicing technique is optimal in the sense that it maximizes the minimum task laxity. The optimality applies to task assignments and communication costs that are completely known *a priori*.

In [26], Gutieérrez García and González Harbor present an approach that derives deadlines for preemptive, deadline-monotonic task scheduling. Given an initial local deadline assignment, the strategy seeks to find an improved deadline assignment using a heuristic iterative approach.

Saksena and Hong present a deadline-distribution approach for pre-allocated tasks in [27] and [28]. The approach specifies the end-to-end deadline as a set of local deadline-assignment constraints, and calculates the largest value of a scaling factor based on a set of local deadline assignments known *a priori*. The scaling factor is then applied to the task execution time. The local deadline assignment is chosen to maximize the largest value of the scaling factor.

In [29], Jonsson and Shin presents a deadline-distribution scheme that distributes task deadlines using adaptive metrics. The authors experimentally show that their scheme yields significantly better performance in the presence of high resource contention. The deadline distribution problem that is addressed in [29] focuses on distributed hard read-time systems with relaxed locality constraints. Thus, schedulability analysis is performed at pre-run-time and only a subset of the tasks are constrained by pre-assignment to specific processors.

Thus, to the best of our knowledge, past efforts on time constraint decomposition has focussed on the deadline constraint. We are not aware of any time constraint decomposition works that consider TUFs.

VIII. Conclusions and Future Work

In legacy environments, time constraints of DTs that are expressed using TUFs can be decomposed for resource-contention resolution and scheduling to improve their timeliness. In this paper, we present methods for decomposing TUFs and identify conditions under which TUF decomposition can improve DT's performance. Using extensive simulation, we show that, in legacy environments, the performance of TUF decomposition is affected by many factors that interact with each other. Among the factors, the most important ones include the properties of node scheduling algorithms, TUF shapes, *task load*, GSF, local threads, and resource dependencies.

There are several interesting directions for future work. One direction is to consider task models with stochastically specified task properties including that for execution times, as they can better model uncertainty. Another interesting direction is to develop distributed scheduling algorithms for scheduling distributable threads.

Acknowledgements

This work was supported by the U.S. Office of Naval Research under Grant N00014-00-1-0549.

References

[1] OMG, "Real-time corba 2.0: Dynamic scheduling specification," OMG Final Adopted Specification, Tech. Rep., September 2001.

[2] "MP radar technology insertion program," http://www.globalsecurity.org/intell/systems/mp-rtip.htm/.

[3] "Bmc3i battle management, command, control, communications and intelligence," http://www.globalsecurity.org/space/systems/bmc3i.htm/.

[4] E. D. Jensen, "Asynchronous decentralized real-time computer systems," in *Real-Time Computing*, ser. Proceedings of the NATO Advanced Study Institute, Springer Verlag, October 1992.

[5] P. Li, B. Ravindran, H. Wu, and E. D. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Transactions on Computers*, submitted August 2003 (under review). Available at: http://nile.ece.vt.edu/submissions/GUS-TOC03.zip.

[6] C. D. Locke, "Best-effort decision making for real-time scheduling," Ph.D. dissertation, Carnegie Mellon University, 1986, CMU-CS-86-134.

[7] R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, CMU, 1990, CMU-CS-90-155.

[8] G. Koren and D. Shasha, "D-over: An optimal on-line scheduling algorithm for overloaded real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1992, pp. 290–299.

[9] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," *Journal of Real-Time Systems*, vol. 10, no. 3, pp. 293–312, May 1996.

[10] D. Mosse, M. E. Pollack, and Y. Ronen, "Value-density algorithm to handle transient overloads in scheduling," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, June 1999, pp. 278–286.

[11] J. Wang and B. Ravindran, "Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis," *IEEE Transactions on Parallel and Distributed Systems*, accepted August 2003, To appear, Available at http://nile.ece.vt.edu/.
[12] OMG, "Real-time corba 1.0, joint submission," OMG Document orbos/1998-12-05, Tech. Rep., 1998.
[13] R. Clark, E. D. Jensen, and et al., "An adaptive, distributed airborne tracking system," in *Proc. of The 7th WPDRTS*, ser. Lecture Notes in Computer Science, vol. 1586. Springer-Verlag, April 1999, pp. 353–362.
[14] D. P. Maynard, S. E. Shipman, R. K. Clark, and et al., "An example real-time command, control, and battle management application for alpha," CMU C.S. Dept., Archons Project TR-88121, Dec. 1988.
[15] The Open Group Research Institute's Real-Time Group, *MK7.3a Release Notes.* Cambridge, Massachusetts: The Open Group Research Institute, October 1998.
[16] D. Wells, "A trusted, scalable, real-time operating system," in *Proceedings of The Dual-Use Technologies and Applications Conference*, 1994, pp. 262–270.
[17] E. D. Jensen, "Private communication," 2003.
[18] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[19] S. Oh and S. Yang, "A modified least-laxity-first scheduling algorithm for real-time task," in *Proceedings of the 5th International Conference on RTCSA*, Hiroshima, Japan, October 1998, pp. 31–36.
[20] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1268–1274, December 1997.
[21] A. Varga, OMNET++ Community Site, http://www.omnetpp.org/.
[22] R. Bettati and J. Liu, "Algorithms for end-to-end scheduling to meet deadlines," in *Proceedings of The Second IEEE Conference on Paralle and Distributed Systems*, 1990.
[23] ——, "End-to-end scheduling to meet deadlines in distributed systems," in *Proc. of RTSS*, 1992, pp. 452–459.
[24] B. C. Kao, "Scheduling in distributed soft real-time systems with autonomous components," Ph.D. dissertation, Princeton University, November 1995.
[25] M. D. Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," in *Proceedings of IEEE Real-Time Systems Symposium*, 1994, san Juan, Puerto Rico.
[26] J. J. G. García and M. G. Harbor, "Optimized priority assignment for tasks and mesages in distributed hard real-time systems," in *Proceedings of WPDRTS*, Santa Barbara, California, 1995, pp. 124–132.
[27] M. Saksena and S. Hong, "An engineering approach to decomposing end-to-end delays on a distributed real-time system," in *Proceedings of WPDRTS*, Honolulu, Hawaii, 1996, pp. 244–251.
[28] ——, "Resource conscious design of distributed real-time systems: An end-to-end approach," in *Proceedings of the IEEE Intl. Conf. on Engineering of Complex Computer Systems*, Montreal, Canada, 1996, pp. 306–313.
[29] J. Jonsson and K. G. Shin, "Robust adaptive metrics for deadline assignment in distributed hard real-time systems," *Real-Time Systems*, vol. 23, no. 3, pp. 239–271, November 2002.

Appendix

A. Additional Simulation Results for GUS and LBESA with Different TUFs

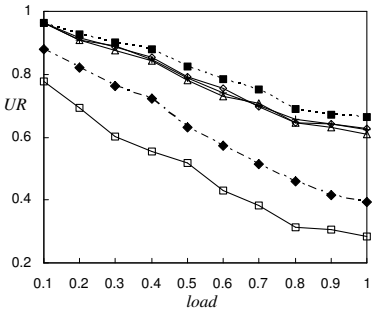
We give simple descriptions and indicate similar curves of the figures in Table V.

TABLE V
Similar Curves in Figures of Appendix A

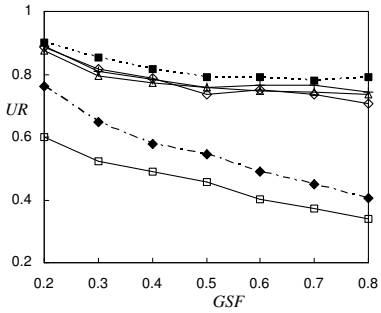
Figure 32	$UR_{GUS}^{SLEQF}(TUF_2) = UR_{GUS}^{SLALL}(TUF_2) \approx UR_{GUS}^{SCEQF}(TUF_2) = UR_{GUS}^{SCALL}(TUF_2)$
Figure 34	$UR_{LBESA}^{SLEQF}(TUF_2) = UR_{LBESA}^{SLALL}(TUF_2) = UR_{LBESA}^{SCEQF}(TUF_2) = UR_{LBESA}^{SCALL}(TUF_2)$. The other curves are similar to each other
Figure 35	Showing the average performance of TUFs—$UR(TUF_5)$ $UR_{LBESA}^{UT} \approx UR_{LBESA}^{OPTCON} \approx UR_{LBESA}^{OPTLNR} \approx UR_{LBESA}^{TUFS} \approx UR_{LBESA}^{STEPS}$ $UR_{LBESA}^{SLEQF} \approx UR_{LBESA}^{SCEQF}$; $UR_{LBESA}^{SLALL} \approx UR_{LBESA}^{SCALL}$.

B. Additional Simulation Results for GUS with Dependencies

We give simple descriptions and indicate similar curves of the figures in Table VI.

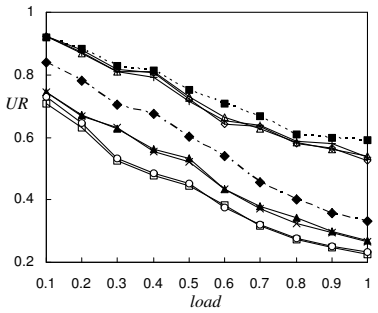


(a) UR vs. $load$

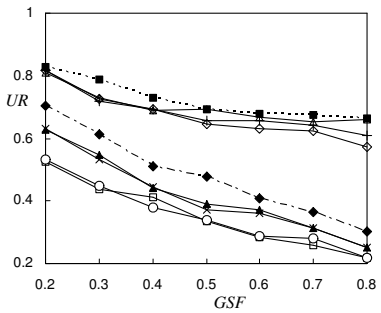


(b) UR vs. GSF

Fig. 32. Utility Ratio with GUS under TUF_2

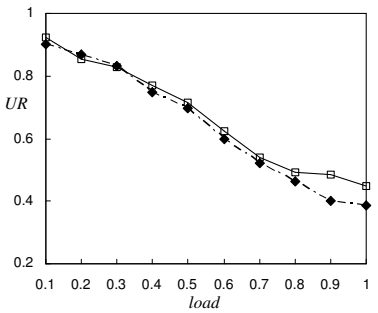


(a) UR vs. $load$

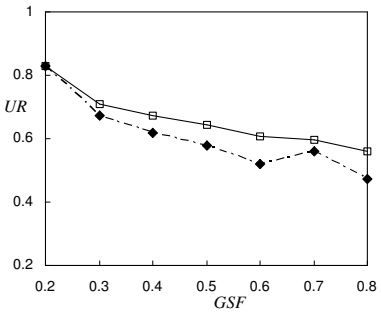


(b) UR vs. GSF

Fig. 33. Utility Ratio with GUS under TUF_5



(a) UR vs. $load$



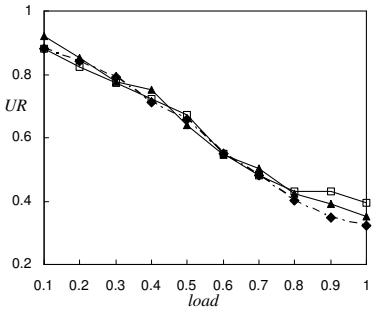
(b) UR vs. GSF

Fig. 34. Utility Ratio with LBESA under TUF_2

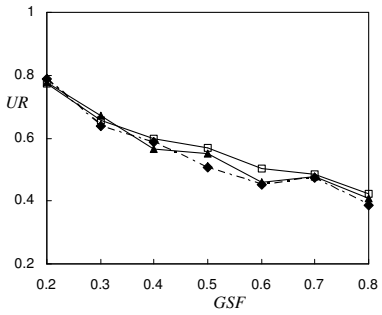
TABLE VI

Similar Curves in Figures of Appendix B

Figure 36, 37	$UR_{GUS}^{SLEQF}(TUF_{1/2}) = UR_{GUS}^{SLALL}(TUF_{1/2})$, $UR_{GUS}^{SCEQF}(TUF_{1/2}) = UR_{GUS}^{SCALL}(TUF_{1/2})$
Figure 36(b), 37(b)	$UR_{GUS}^{OPTCON}(TUF_{1/2}) \approx UR_{GUS}^{OPTLNR}(TUF_{1/2})$
Figure 40	$UR_{GUS}^{SLEQF}(TUF_{1/2}) = UR_{GUS}^{SLALL}(TUF_{1/2})$, $UR_{GUS}^{SCEQF}(TUF_{1/2}) = UR_{GUS}^{SCALL}(TUF_{1/2})$ $UR_{GUS}^{TUFS}(TUF_{1/2}) \approx UR_{GUS}^{STEPS}(TUF_{1/2})$

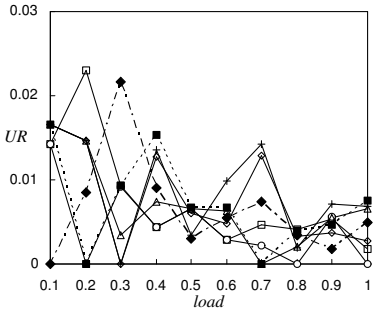


(a) UR vs. $load$

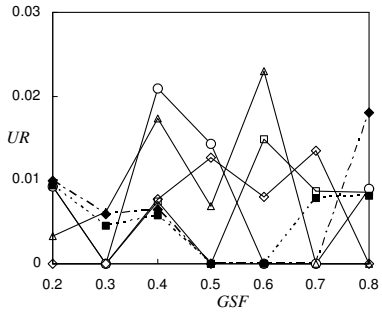


(b) UR vs. GSF

Fig. 35. Utility Ratio with LBESA under TUF_5

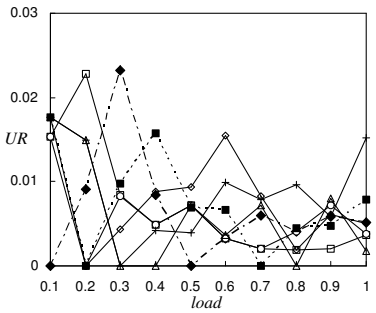


(a) UR vs. $load$

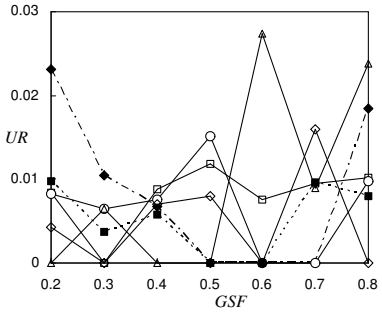


(b) UR vs. GSF

Fig. 36. Utility Ratio with GUS and Resource Dependencies under TUF_1

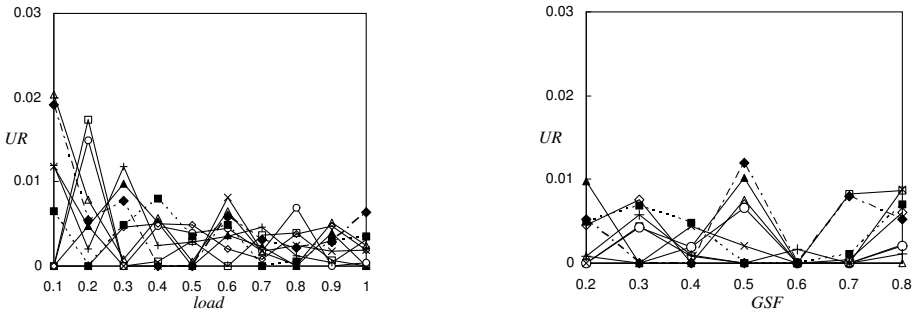


(a) UR vs. $load$



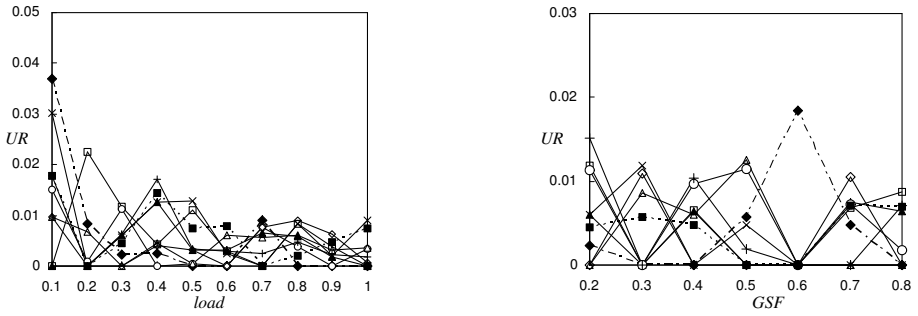
(b) UR vs. GSF

Fig. 37. Utility Ratio with GUS and Resource Dependencies under TUF_2



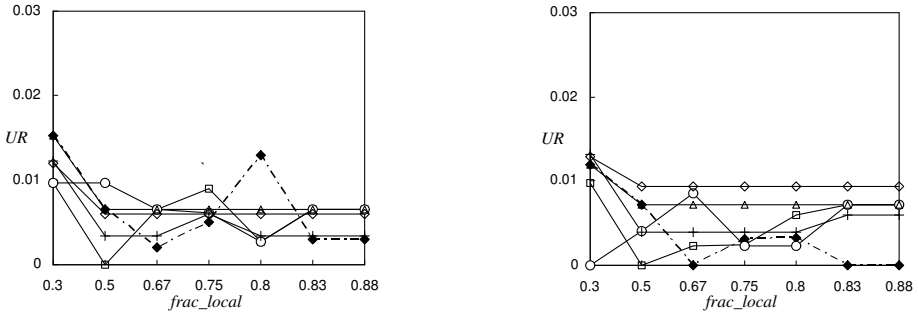
(a) UR vs. $load$ (b) UR vs. GSF

Fig. 38. Utility Ratio with GUS and Resource Dependencies under TUF_4



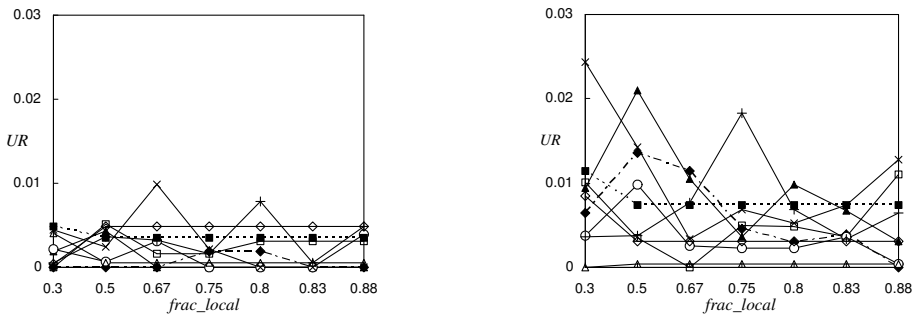
(a) UR vs. $load$ (b) UR vs. GSF

Fig. 39. Utility Ratio with GUS and Resource Dependencies under TUF_5



(a) UR vs. $frac_local$ under TUF_1 (b) UR. vs. $frac_local$ under TUF_2

Fig. 40. UR vs. $frac_local$ with GUS and Resource Dependencies, $TUF_{1/2}$



(a) UR vs. $frac_local$ under TUF_4 (b) UR. vs. $frac_local$ under TUF_5

Fig. 41. UR vs. $frac_local$ with GUS and Resource Dependencies, $TUF_{4/5}$