Ensuring Schedulability in Large-Scale Distributed Real-Time Systems
(Position Paper)

Riccardo Bettati  Wei Zhao
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
{bettati,zhao}@cs.tamu.edu

Abstract

As large-scale distributed real-time systems become support a variety of dynamic applications and many paradigms for interaction among their components, the development and analysis methodologies to support the design and the verification of these emerging systems has not kept place. In this position paper we focus on the need for schedulability analysis, flexible resource modeling and system design in support of schedulability analysis for distributed real-time systems. We will propose an integrated analysis methodology that is applicable to a variety of system design paradigms, in particular for emerging component-based systems.

1 Introduction

In the last few years, the following pattern has become apparent for distributed real-time embedded (DRE) systems:

- With the current tendency towards COTS components the computation and communication infrastructure is becoming more and more heterogeneous, in terms of programming languages, operating system support, end-systems, and networking technologies.

- Such systems are supporting a variety of paradigms for the interaction among their components. This ranges from traditional stream-based communication, where data traverses the system in well-defined flows, to publish-subscribe approaches, to highly dynamic remote method invocation; in two-party as well as multi-party settings.

- There is a need for integrated support for timely delivery of service and for reliability and security. Real-time group communication, for example, is and will be applied in various forms to realize replication.

- There is a strong need for the ability to build such systems by integrating reusable software components. A large number of efforts are under way to provide (i) a standard software infrastructure, often in form of CORBA middleware, to enable the re-use of software components in a "plug-and-play" fashion and (ii) system layer services for adapting to changes in the application requirements and in the environment.

In this paper, we address three issues in the design, realization, and operation of large-scale DRE systems: (1) dynamic end-to-end delay guarantees, (2) integration of a management framework for QoS provisioning, and (3) the underlying models and methodologies for efficient verification of real-time guarantees.

2 Dynamic End-to-End Delay Guarantees

For mission critical applications in DRE systems, meeting the end-to-end deadlines is a critical concern. Tools are therefore necessary to verify whether or not end-to-end deadlines can be met. Such tools should be applicable during design time for design verification as well as during runtime when adaptation is to take place.

Realization of effective and scalable schedulability testing tools is particularly challenging as the development of methodologies to support the verification of delay properties for these emerging DRE systems has not kept pace with other key functions. This lag has been caused by a number of reasons; we elaborate on some of them below:

Traditional methodologies lack an integrated model for computation and communication. Practical considerations have traditionally led to very different ways of analyzing real-time computation and real-time communication. For example, the comparatively small granularity of units of
workload in networks allows for round robin scheduling (and variations thereof) in switches and routers. These classes of schedulers are less appropriate in computation nodes because units of computation are rather large. Similarly, network elements, (such as switches and routers) exhibit little or no blocking. Semaphores and other synchronization mechanisms in the end-systems can cause complicated blocking behavior with ensuing priority inversions. This artificial separation of computation and communication is awkward (for example, it leads to lower utilizations), and it becomes more so as the boundary between the two becomes blurred, as sophisticated communication primitives (e.g., reliable group communication) evolve.

Traditional methodologies make simplistic assumptions about resources. Active resources, such as CPUs or communication links, are typically modeled as idealized constant rate servers. In real systems, the rate at which jobs can be served is highly variable. At CPU level, for example, caches and instruction pipelining cause short-term variations in the perceived processor speed. Lower-level operating system layers add various forms of hidden scheduling and priority inversion [1, 4]. Simply assuming a worst-case rate for processing nodes is a common technique, which unduly reduces system utilization. Rather, a resource model must be found that allows for the explicit description of the worst-case availability of particular jobs.

Traditional methodologies rely heavily on workload regulation. Formulas for delay computation make assumptions about worst-case workload, and various forms of regulators make sure that these assumptions are satisfied. For example, rate controllers in packet schedulers (e.g. [5]) make sure that the packets presented to the scheduler in a switch or a router adhere to a previously specified minimum interarrival time. Similarly, resource access protocols based on priority inheritance or priority ceiling [2, 3] control the eligibility for execution of a portion of a task by appropriately modifying the task’s priority. The pervasive use of regulation is problematic for the class of systems described above, for two reasons:

First, many of the available hardware and software components must be treated as black boxes, with only a few providing hooks for regulation, or even for scheduling control. For example, legacy components may contain their own access resolution schemes to resources such as FIFO queues for name server in distributed networks.

Second, it is generally perceived that regulation adds significant overhead during system operation, be it through the inherent overhead of the regulation or through additional wrappers needed to put regulation into place.

It is therefore necessary to focus on schedulability analysis, flexible resource modeling, and system design for support of schedulability analysis. This in addition to the integration of the resulting methodologies into existing middleware in order to provide the necessary enforcement and monitoring capabilities.

3 Requirements for QoS Provisioning

Schedulability Analysis. Some form of schedulability analysis is performed in every system where timing guarantees must be met. This typically happens at design time for static systems, and as part of an admission control process in systems with dynamically changing workload.

The central component of every schedulability analysis methodology is the computation of worst-case response times (either deterministic or statistical) of the jobs under consideration. Once the worst-case response time has been determined for a particular job, it is compared against the timing requirements to check whether they are met. (Alternatively, schedulability can be determined indirectly, for example by relying on resource utilization.)

Traditionally, work on schedulability analysis in end-systems focuses on periodic jobs, where the inter-arrival time of requests is expected to be the period of the job. Non-periodic workload is typically transformed into a periodic one (either by workload transformation or other regulation at the servers) so that well-known schedulability analysis methodologies for periodic workloads can be used. Applying the same methods for distributed real-time systems, where jobs execute across multiple endsystems, shows poor results, even for periodic workloads.

By appropriately synchronizing, or regulating the execution of the jobs on the processors, excessive bursts can be eliminated, which increases schedulability, and the job execution can be made to adhere to simple workload descriptors, which makes a rigorous schedulability analysis possible. It is a well-known fact that appropriate regulation reduces the worst-case end-to-end response times as compared to systems with no regulation. However, regulation adds overhead to the system and increases the average end-to-end response times for jobs. In addition, it is of limited applicability when the workload as generated by the applications is inherently aperiodic.

Finally, not all components in the system may be accessible enough to allow for integrated regulation. Wrappers will then have to be realized, which may unduly add to system overhead.

To deal with systems that have limited or no support for workload regulation, we have developed a methodology for workload re-characterization. Instead of reshaping the workload after each processor to conform to a predetermined description, we use general workload description functions, which are computed at each processor. The workload descriptors depend on the scheduling policies and the other workload present on the processor. Our group has applied these techniques in various forms at network level,
and the next section gives a short overview on how this can be extended into the endsystems.

It is important to note at this point that we do not envision workload re-characterization as a replacement for regulation. Rather, it is complementary and can be naturally combined with it, allowing for an integrated analysis methodology.

**Flexible Resource Modeling.** A model for processors must reflect that resources are not ideal. (i) Service to particular jobs may be interrupted or delayed because of various forms of priority inversion (be these caused by instruction-level effects or by the operating system, or by higher-level resource accesses). (ii) Processors may be controlled by a variety of different scheduling policies. (iii) A modeling methodology must lend itself to effectively and accurately model hierarchical compositions of processors, be these collections of processors, or processors in combination with operating system layers incorporating resource managers, or combinations thereof.

**Design for Schedulability Analysis.** Traditionally, approaches to provide real-time services in a system are strictly scheduling based. In this spirit, limitations on the underlying operating system are eliminated. While such improvements to the underlying infrastructure reduce response times for time-critical jobs, they are often difficult to realize (in particular in a heterogeneous environment,) difficult to port, and not always necessary. If an analysis shows that timing requirements are met, then deadlines are maintained, whether priority inversion is present in the system or not. To make such an analysis effective, it is important to appropriately plan a system with schedulability analysis in mind. This entails (i) appropriate identification of resources and description of access protocols, (ii) hooks for pre-runtime execution time estimation, and (iii) generic support for placement of monitoring points to support these estimations and for feedback during runtime.

**4 QoS Provision Management Framework**

There are two key problems with current solutions for QoS provisioning mechanism integration. First, existing DRE solutions do not allow for the systematic decoupling of application functional paths from their various cross-cutting QoS properties. Moreover, current DRE models use an ad-hoc approach to the integration of QoS provisioning mechanisms. Together, these shortcomings create a suite of problems (e.g., maintenance headaches, impaired usability, QoS change inflexibility, etc.). The answers to these challenges lie in an approach that mimics the success of current COTS enterprise component architecture (ECA) middleware technologies, but with a twist.

Designed for the business computing world, ECA frameworks (e.g., COM+, EJB, CORBA CCM) systematize service provisioning by allowing key services to be reused and shared, obviating the need for hard-coding the same services in an ad hoc fashion for each application. Core application services (scalability, security, persistence, transactions, asynchronous communications, etc.) are built into the container by the COTS server providers and can be leveraged again and again by developers. However, one key drawback of the ECA approach is its lack of support for real-time systems. Hence, embracing ECA for today’s DRE requirements has, until now, created as many problems as it has solved.

In a typical runtime environment supporting DRE applications, the QoS provisioning mechanisms can be modeled with metadata and meta-components. Meta-programmable DRE middleware has become an important research topic. Though it addresses the challenges of QoS provision management integration, adding real-time capabilities to ECA systems leaves another challenge to be answered - how to add new meta programmable capabilities to existing container architectures.

A realistic pathway for QoS provision mechanism integration is needed, which marries the successes of ECA frameworks with the requirements of designers and operators of distributed real-time and embedded systems. A QoS provision management framework for the systematic creation and management of new components can be added to existing container-based suites of component services. Appropriate mechanisms for negotiation, adaptation, and enforcement and monitoring must be in place in order to achieve this objective.

The **basic requirements** for a QoS guarantee system are as follows:

**Scalable:** Admission control and packet forwarding scheme must be scalable regardless of the number of flows in the system. This means that the overhead of admission control and packet forwarding is independent of the number of flows in the system.

**Effective:** Admission control must maximize the bandwidth utilization to the extent possible. This means that it must be highly accurate despite the lack of per-flow information.

**Adaptive:** Resource allocation has to be cognizant of the dynamic fluctuations in resource availability. This will lead to a better quality of services and better utilization of system resources.

**Compatible:** For practical purposes, the system must be compatible with current industrial practice.

In order to provide service guarantees in a scalable and
Effective manner, a two-module mechanism is required, in addition to any monitoring and adaptation:

**QoS Verification and Planning Module:** This module comes into play during initial system configuration, or during modification of service level agreement. This phase consists of two interleaved functions:

**Resource planning:** Various plans are made for resource allocation in accordance to different run-time scenarios. Various means of resource allocations to different components are considered such priority assignments, bandwidth allocation, etc.

**Delay verification:** Here, we propose to use our novel task-population-insensitive delay calculation method. With method, we can verify whether the end-to-end delay bound in distributed system satisfies the deadline requirement as long as the utilization of resources is within a pre-defined limit.

**Admission Control Module:** This module is invoked when a new task flow is requested. Admission control module makes sure that enough sources are available to satisfy the requirements of both the new and the existing task flows after the new task has been admitted. In order to meet scalability requirement, any admission control mechanism must be lightweight so that it can be realized in a scalable fashion. Per the function of our verification and planning module, the delay guarantee test at run time is reduced to a simple utilization-based test: As long as the utilization of hosts and links along the path of a flow is not beyond a given bound, the performance guarantee can be met. Recall that the value of the utilization bound is verified at system (re-)verification and planning time. Once verified, the use of this utilization bound is relatively simple at flow admission time: Upon the arrival of a task flow request, the admission control admits the task flow if the utilization values of links along the path of the new task flow are no more than the bound. Thus, this approach (called Utilization-Based Admission Control - UBAC - in the following) eliminates explicit delay analysis at admission time, and renders the system scalable.

### 5 Derivation of Utilization Bounds

#### 5.1 Workload Modeling

We elaborated earlier on the critical role that workload modeling in delay-guaranteed distributed computation. Only if we properly model the task flow traffic in the network, we can efficiently derive tight worst-case delay bounds and hence provide delay guarantees to mission critical applications.

Traditionally, periodic and stochastic models have been used for characterizing network traffic [11]. However, the stochastic model only provides insight into the average performance of the network that is not adequate for mission critical real-time systems. On the other hand, while the periodic model has been used for single-node real-time systems, it is difficult to apply to general distributed systems, as a periodic task flow would no longer be periodic after first node.

To provide delay guarantees in distributed embedded systems, we seek mathematical tools that can describe bounds on the behavior of task flows everywhere in the system. For example ([8, 12],) the following mathematical functions characterize the task flows from a source at an arbitrary point of the system:

\[
F(I) = \max_{t}(A(t + I) - A(t))
\]

and

\[
f(I) = \min_{t}(A(t + I) - A(t))
\]

where \(A(t)\) is the number of tasks that have arrived by time \(t\), \(A(t + I) - A(t)\), thus, describes the number of tasks arrived during time interval \((t, t + I)\). Consequently, \(F(I)\) and \(f(I)\) define the maximum and minimum numbers of tasks that arrive during any interval of length \(I\), respectively. In the case that a task flow is constant-rated, then

\[
F(I) = f(I) = \alpha I
\]

where \(\alpha\) is the rate.

Using these functions, we developed various methods to predict the worst-case delays of tasks and messages, as required by our previous real-time system projects [7, 8]. We briefly overview the major relevant results as follows:

- These task-flow modeling functions can describe a wide variety of real-time task sources [8]. Different sources generate different kinds of task flows. For example, the flow generated by a multi-media application may be very different from that generated by a process monitoring and control application.
- Cost effective representation of these traffic functions can be obtained [13]. This is critical in improving the efficiency of admission control module.
- These traffic functions can be analytically derived [6]. Let \(F_{\text{in}}, F_{\text{out}}, f_{\text{in}},\) and \(f_{\text{out}}\) be the maximum and minimum task-flow functions at the input and output of a host, respectively. Then, the following formulas can be proved:

\[
F_{\text{out}}(I) = F_{\text{in}}(I + d)
\]

and

\[
f_{\text{out}}(I) = f_{\text{in}}(I - d)
\]
where $d$ is the worst case delay at the host. The value for $d$ depends on the scheduling methodology used in the host and can be obtained by various analytical techniques. Formulas (4) and (5) imply that the task flow at the output of a host, modeled by $F_{out}$ and $f_{out}$, can be derived from the task flow at its input (i.e., the output of the previous host). Consequently, task flow functions along the entire path can be derived once the source flow is specified.

These observations, especially the third one, are particularly important for deriving delay bounds, as we will see below.

### 5.2 Delay Derivation and Dependency Removal

A major advantage of using the above task flow functions is that the delay be derived relatively easily. Substantial literature exists for delay derivations with different scheduling algorithms. We illustrate the methodology by an example of system with static-priority schedulers. Consider a host serving a set of task flows. Let $F_{i,j}$ be the maximum task flow function for the $i$-th task flow with priority $j$. Let its worst case computation time be $C_{i,j}$. Let $d_j$ be the worst case waiting time of tasks with priority $j$.

Let $t$ be the starting time of a busy interval. Assume that at time $t+I$ a task instance from task flow $T_{i,j}$ arrives. How long will this task wait?

First, this task has to wait for all the priority-$j$ tasks that arrived before it. Between $t$ and $t+I$, for tasks with priority $j$, the demand (i.e., the total requested computation time) is given by

$$\sum_{i} F_{i,j}(I)C_{i,j}$$

Second, this task has to wait for all tasks with priorities higher than $j$. In the worst case, at the time this task being served, the demand of these high priority tasks will be given by

$$\sum_{k=1}^{i-1} \sum_{i} F_{i,j}(I + d_j)C_{i,j}$$

However, by $t+I$, the host has served tasks for $I$ units of time. Thus, the total unfinished work in front of the newly arrived task of priority $j$ is given by

$$\text{sum}_i F_{i,j}(I)C_{i,j} + \sum_{k=1}^{j-1} \sum_{i} F_{i,j}(I + d_j)C_{i,j} - I$$

Consequently, the worst case waiting time for a task with priority $j$ is given by

$$d_j = \max_{t>0} (\sum_{i} F_{i,j}(I)C_{i,j} + \sum_{k=1}^{j-1} \sum_{i} F_{i,j}(I + d_j)C_{i,j} - I)$$

This example shows that by using the maximum task flow function, it is relatively straightforward to derive the worst-case delay. The derivation for other scheduling algorithms is similar.

In any case, while we can solve the worst-case delay from Equation (9), the delay according to (9) depends on task worst case computation time and the number of tasks associate with each priority. To realize a utilization-based end-to-end delay guarantee, we need to remove these dependencies.

To remove the worst-case computation time, we can simply unitize the tasks. That is, a task with the worst-case computation time $C$ can be equivalently transformed to $C$ unit-size tasks. After this transform, the delay will only depend on the number of tasks in the host.

In [9, 10] we developed techniques to remove the dependency on the number of tasks. For example, if a host has two priorities and the utilization of high priority tasks is no more than $\alpha$, then the worst-case end-to-end delay of the high priority tasks is given by

$$d \leq \frac{1}{\alpha} - (h - 1) \rho$$

where $h$ is the maximum number of hosts a flow may cross. $\sigma$ and $\rho$ are the burstiness and long-term rate of high priority tasks, respectively. Both are usually parameters of task flow functions and can be measured. Formula (10) is readily used in admission control. As $\sigma$ and $\rho$ are given, the admission control module just needs to obtain the current utilization on each host, substitute it into (10) to obtain an upper bound of the worst-case end-to-end delay. While the formulas of other cases may be more complicated than (10), the results are similar.

### 5.3 Statistical Guarantees

While the utilization-based schedulability testing has been approved as an effective technology, several extensions are needed in order to adopt it into the domain of large-scale environments with dynamic QoS provisioning for new generation mission critical applications.

In many situations, a deterministic delay guarantee is not necessary. To fully utilize the system resource, we may only need to guarantee certain percentage of tasks to meet their deadlines. Our schedulability testing needs to be extended to take this factor into account. Several approaches are possible. The following are some initial approaches we may take.

One way to deal with this is to let $F()$ be a statistical one. Consequently (with certain statistical and analytical
manipulations), we may obtain Formula (10) in a probabilistic form. That is, we want to derive a delay upper bound that only holds with certain probability.

Another method is to randomize $\alpha$ itself. That is, we will consider the current utilization is a random variable with certain distribution. We need to re-derive (10) to make it reflect the distribution of $\alpha$. Once we do that, we can use the new (10) as we do in the deterministic case.

6 Summary

In this paper, we argue for the need for integrated approaches for schedulability analysis that take into consideration the needs to (1) model and analyze communication and computation subsystems in an integrated fashion, (2) allow for scalable and light-weight re-validation of performance guarantees during re-configuration and adaptation for large number of components and workloads, and (3) allows for hierarchical representation and for aggregation of both workload and system resources and components.

We propose flexible workload and resource modeling through arrival and service functions and appropriate re-characterization of workload within the system to effectively manage end-to-end real-time guarantees. We propose an extension to traditional utilization-based schedulability analysis that allows for very light-weight resource management in distributed systems.

Successful management of large-scale systems at the level required to provide deterministic guarantees remains an elusive goal. Scalable approaches must be found that integrate the analysis of controllable subcomponents and deterministic workload with those subsystems and workload that cannot be accurately controlled. We are laying out a plan to investigate utilization-based schedulability analysis in mixed environments, where deterministic and stochastic subsystems co-habit, and deterministic bounds on resource availability and workload can be guaranteed for portions of the system.

References


