Online Robust Optimization Framework for QoS Guarantees in Distributed Soft Real-Time Systems

Jinkyu Lee
Dept. of Computer Science
KAIST, South Korea
jinkyu@cps.kaist.ac.kr

Insik Shin[*]
Dept. of Computer Science
KAIST, South Korea
insik.shin@cs.kaist.ac.kr

Arvind Easwaran
CISTER Research Unit
Polytechnic Institute of Porto, Portugal
aen@isep.ipp.pt

ABSTRACT

In distributed soft real-time systems, maximizing the aggregate quality-of-service (QoS) is a typical system-wide goal, and addressing the problem through distributed optimization is challenging. Subtasks are subject to unpredictable failures in many practical environments, and this makes the problem much harder. In this paper, we present a robust optimization framework for maximizing the aggregate QoS in the presence of random failures. We introduce the notion of K-*failure* to bound the effect of random failures on schedulability. Using this notion we define the concept of K-*robustness* that quantifies the degree of robustness on QoS guarantee in a probabilistic sense. The parameter K helps to trade-off achievable QoS versus robustness. The proposed robust framework produces optimal solutions through distributed computations on the basis of Lagrangian duality, and we present some implementation techniques. Our simulation results show that the proposed framework can probabilistically guarantee sub-optimal QoS which remains feasible even in the presence of random failures.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

General Terms

Algorithms, Design, Performance, Theory

Keywords

Soft Real-Time Systems, Robust Optimization, QoS Guarantee

[*]A corresponding author

1. INTRODUCTION

Distributed real-time systems are typically comprised of several nodes with processing capacity and several real-time tasks that execute on these nodes. Each task, in turn, comprises of several subtasks that have processing demands at nodes and are required to execute in a pipelined fashion. These tasks often require guarantees on end-to-end delays, with smaller delay implying a better quality-of-service (QoS). For example, we consider delay-sensitive tasks such as security monitoring and video conferencing in enterprise security environment [3]. These tasks can be expressed as a set of sequential subtasks in that sensor nodes monitor their circumference and send raw data to their destination nodes through a set of intermediate nodes, which process (filter and analyze the data, update database [17]) and relay the data, for the purpose of security or enterprise.

Soft real-time constraints can be conveniently modeled using delay-sensitive utility functions (see for example [28]). Maximizing the collective utilities of tasks (aggregate QoS) is then a reasonable system-wide goal, and achieving this inherently brings the tasks into resource contention with each other. The end-to-end delay of a task may be reduced by finishing its subtasks earlier, but this increases interference to subtasks of other tasks that compete for processing capacity on the same nodes. The end-to-end delay of these other tasks may then increase, thereby reducing their utility. It is therefore necessary to investigate the effects of finishing a subtask earlier, both on end-to-end delay of the corresponding task (across nodes) and on other subtasks that are scheduled within the same node (across tasks). In other words, maximizing the system-wide QoS inherently entails a global optimization approach.

In this paper, we control the finishing time of subtasks by assigning local (artificial) deadlines to them and enforcing these deadlines through node-level schedulability tests. We introduce a QoS optimization framework that uses global viewpoints to assign local subtask deadlines (across nodes), taking into consideration node-level schedulability (across tasks). Further, to benefit from many advantages of distributed computation such as scalability, we develop a distributed optimization framework based on Lagrangian duality theory [6, 19]. This framework guarantees that the online distributed computations eventually converge to a global optimum under certain conditions, including a failure-free assumption. However, in many practical environments, subtasks are subject to random failures, such as transient hardware faults and network medium errors. These failures embed uncertainty into the timing attributes of subtasks; subtasks may need to be re-executed in order to recover from these faults. Improper handling of such uncertainty may result in fluctuations in the achieved system utility.

Further, this unstable behavior may also invalidate the guarantee on QoS provided by the optimization framework.

In this paper, our goal is to develop an online "robust" distributed optimization framework that can provide guaranteed QoS under random failures. In order to develop such a framework, we address quite a few challenges: 1) how to derive the uncertainty set that characterizes the effect of random failures on the ability of subtasks to meet local deadlines (node-level schedulability), 2) how to define a notion of robustness against the failures in these systems, and 3) how to incorporate this notion in the distributed optimization framework, without violating the conditions imposed by Lagrangian duality theory (namely, convexity of optimization constraints).

Contribution. The contributions of this paper are as follows. We present a robust and distributed optimization framework that guarantees QoS always stays over a lower bound, and this guarantee remains valid (and therefore the framework is robust) in a probabilistic sense even in the presence of random failures. To the best knowledge of the authors, this is the first such framework for optimizing the aggregate QoS in distributed real-time systems. Specifically, we introduce the notion of K-*failure* (at most K concurrent failures among all subtasks in a node) to bound the effect of random failures on node-level schedulability. We also introduce a notion of K-*robustness* to quantify the degree of robustness against such failures with a probability. This parameter K offers a tradeoff between QoS guarantee and probabilistic robustness. Using this new metric for robustness, we derive a robust optimization formulation that is amenable to online distributed computations (Lagrangian duality). Further, the node-level schedulability tests in this framework are *sustainable* [4] with respect to the execution time of subtasks. That is, the solution remains valid even if the actual execution time of a subtask is smaller than its stated worst-case execution time or the actual number of failures in a node is smaller than K. Lastly, we discuss some implementation issues for the framework, and evaluate its performance.

Related Work. Many studies have focused on the local subtask deadline assignment problem, with a view to controlling end-to-end delays [12, 21, 16]. These studies focus on how to divide the deadline of a task into several pieces for its subtasks, but they have lack of considering the resource contention in intermediate nodes between subtasks from different tasks. There are some studies on end-to-end delay analysis of distributed real-time systems [14, 15]. These studies focus on reducing the pessimism in calculating end-to-end delays for pipelined streams of computations, but do not consider occurrence of failure or maximizing QoS.

Convex optimization theory has been a popular tool to solve many global optimization problems for several decades. Techniques that find optimal solutions either in a centralized manner, or using distributed computations (Lagrangian duality), have been developed [6, 19]. Many of these techniques have been applied to solve the problem of guaranteeing end-to-end delays in a distributed real-time system [10, 7, 31, 20]. Some of these techniques use centralized solutions to the optimization problem [10, 7, 31]. In addition to these techniques not scaling well in a distributed system, either they are not robust to task failures [10, 7] or they provide only heuristic solutions [31]. Distributed solutions to this optimization problem have been recently considered [20]. This study assumes proportional share scheduling within nodes. Since such a scheduling framework is not implementable, it must be approximated. However, any approximation of the scheduling framework will invalidate the proposed analysis. Distributed optimization has been applied to achieving QoS maximization through dynamic route and rate assignments in distributed real-time systems [24] and through bandwidth allocation in wireless networks [13]. However, this study does not consider guaranteeing end-to-end delays and robustness.

Robustness to variability in task worst-case execution time (WCET) was first introduced in the context of fixed-priority preemptive uniprocessor scheduling [11]. More recently, robustness to uncertain input parameters was considered in a distributed optimization problem [29]. Since this work does not consider how to map uncertainty to node-level schedulability, it cannot be used to bound end-to-end delay of real-time tasks. Therefore it cannot provide guaranteed QoS for systems that are considered in this work. Feedback control techniques have been used to control node utilization against uncertain WCET in distributed real-time systems [27]. Such approaches aim at enhancing system survivability, but do not provide any guarantees on end-to-end delay of tasks.

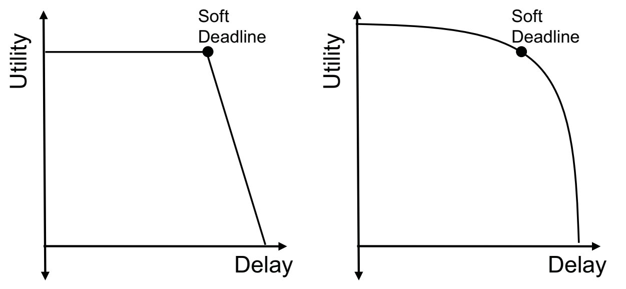
Organization. Our paper is organized as follows: Section 2 describes the system model. Section 3 provides our distributed QoS optimization framework, and Section 4 presents its robust counterpart. Section 5 discusses some practical issues for implementation, and Section 6 presents simulation results. Section 7 concludes with future work.

2. SYSTEM MODEL

2.1 Task model

In this paper, we consider a distributed real-time system with V_N nodes and V_T tasks. The nodes are numbered $1, \dots, V_N$ such that each node has a unique number, and we express a node numbered n as N_n. Each task $\tau_i \in \mathcal{T}_{sys}$ is comprised of m_i subtasks such that each subtask executes on exactly one node. The k^{th} subtask executing on N_n is denoted as $\mathcal{J}_{(i,k,n)}$; whenever n is irrelevant we omit the third parameter completely. Adjacent subtasks $\mathcal{J}_{(i,k)}$ and $\mathcal{J}_{(i,k+1)}$ execute in sequence in a pipe-lined fashion; $\mathcal{J}_{(i,k+1)}$ becomes ready for execution when $\mathcal{J}_{(i,k)}$ completes. We denote the worst-case (maximum) execution time of subtask $\mathcal{J}_{(i,k)}$ by $C_{(i,k)}$. Each task τ_i is a sporadic task such that its first subtask $\mathcal{J}_{(i,1)}$ is released repeatedly with a minimum gap of T_i time units.

Let $d_{(i,k)}$ denote the maximum local delay (or response time) that subtask $\mathcal{J}_{(i,k)}$ experiences in its node; it is a time duration from an instant at which it is released in the node to another instant at which it finishes its execution. Then we denote the end-to-end delay of a task τ_i by d_i, where $d_i = \sum_{k=1}^{m_i} d_{(i,k)}$. We assume that each task has its own utility function U_i, which is a function of its end-to-end delay d_i. Utility functions can be viewed as characterizing different QoS levels. We consider concave and non-increasing utility functions to capture that a greater QoS comes with a shorter end-to-end delay and degradation of QoS gets more severe as delay gets longer. The concave function is particularly good at capturing a situation that degradation of QoS is smooth before a certain point (*i.e.*, a soft deadline), but becomes rapid after this point, as shown in Figure 1. Typical examples of tasks subject to such soft deadlines include plot correlation and track maintenance of a coastal air defense system [28]. In this paper, we consider utility functions to be differentiable in order to incorporate them into the proposed optimization framework. If an original function is not differentiable as shown in Figure 1(a), it can be approximated as the one shown in Figure 1(b). We then define the system utility as



(a) The function capturing a soft deadline (b) The differentiable function

Figure 1: Non-increasing concave utility functions

$$U_{sys} = \sum_{\tau_i \in \mathcal{T}_{sys}} U_i(d_i). \tag{1}$$

Given a set of QoS-sensitive tasks as above, informally, our aim is to provide QoS guarantees as much as possible. We capture it by maximizing the system utility function U_{sys}. Then our goal is to bound the delay d_i of each task $\tau_i \in \mathcal{T}_{sys}$ that maximizes U_{sys}. However, computing this bound exactly in general distributed systems is computationally intractable [5, 22]. Hence we approximate the bound on d_i as follows. For each subtask $\mathcal{J}_{(i,k)}$, we define a local (artificial) deadline $D_{(i,k)}$ such that $D_{(i,k)} \leq T_i$. We derive conditions which guarantee that every occurrence of each subtask finishes by its local deadline, *i.e.*, we enforce the condition $d_{(i,k)} \leq D_{(i,k)}$ for each subtask $\mathcal{J}_{(i,k)}$. Then we can upper bound d_i as

$$d_i = \sum_{k=1}^{m_i} d_{(i,k)} \leq \sum_{k=1}^{m_i} D_{(i,k)}. \tag{2}$$

Thus the problem of bounding d_i for each task τ_i that maximizes U_{sys}, is transformed to the problem of finding $D_{(i,k)}$ for each subtask $\mathcal{J}_{(i,k)}$ that maximizes U_{sys}. In the Section 2.3, we derive conditions that enforce $d_{(i,k)} \leq D_{(i,k)}$ for each subtask $\mathcal{J}_{(i,k)}$.

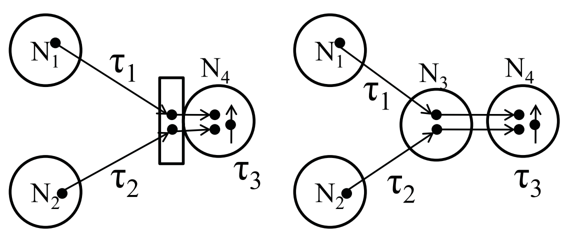
Note that it is essential to decompose the end-to-end delay into delays (and therefore artificial deadlines) for individual subtasks, because of the limitations of existing real-time scheduling theory[1]. These local deadlines enable us to optimize the global system utility while still maintaining the schedulability of individual nodes.

2.2 Scheduling model

In distributed real-time systems, there are two kinds of resource scheduling to consider: scheduling within nodes (CPU scheduling) and network scheduling across nodes. We consider prioritized, preemptive EDF scheduling for nodes, as it is an optimal dynamic-priority uniprocessor scheduler [18].

Network scheduling involves transmission of tasks from one node to another, enforcing the sequential constraint (pipelined execution) between successive subtasks. When a subtask $\mathcal{J}_{(i,k,n)}$ finishes its execution on node N_n, it generates a network subtask that must be transmitted over the network to another node N_o. This network subtask signals the release of subtask $\mathcal{J}_{(i,k+1,o)}$ in node N_o and may also carry data from subtask $\mathcal{J}_{(i,k,n)}$ to subtask $\mathcal{J}_{(i,k+1,o)}$. Since network subtasks generated by many different tasks may be simultaneously transmitting to node N_o, there is a need to schedule these subtasks. This network scheduling problem can be trivially mapped to a node scheduling problem [24] as shown in Figure 2. In the figure, two tasks τ_1 and τ_2 generate network subtasks that may simultaneously transmit to node N_4. This can be viewed as scheduling of a virtual network node N_3 as shown in Figure 2(b). The transmission times of network subtasks generated by τ_1 and τ_2 can then be considered as their execution times on node N_3. Since network scheduling is atomic in practice, we assume that node N_3 employs prioritized, non-preemptive EDF scheduling. Thus the general distributed scheduling problem can be expressed as the problem of scheduling a set of nodes, where each node uses either a preemptive or a non-preemptive EDF scheduler.

[1]Current real-time scheduling theories are mostly developed for node-level schedulability analysis, and the system-level analysis is generally achieved by assembling individual node-level analysis results. Therefore, it is hard to directly support a task model in which 1) a series of subtasks sequentially go though multiple nodes with one end-to-end deadline, and 2) the sets of nodes which subtasks of different tasks pass through are different.



(a) An original task set (b) A unified task set

Figure 2: Representation of a task set

2.3 Node schedulability condition

We now consider a node N_n with subtasks $\{\mathcal{J}_{(i,k,x)} | x = n\}$, and derive schedulability conditions under preemptive and non-preemptive EDF schedulers, *i.e.*, conditions which guarantee $d_{(i,k)} \leq D_{(i,k)}$. Consider the following classical definition of demand bound function that bounds the worst-case (maximum) resource demand of all subtasks in node N_n.

$$dbf_{(i,k)}(t) = \left\lfloor \frac{t + T_i - D_{(i,k)}}{T_i} \right\rfloor C_{(i,k)} \tag{3}$$
$$dbf_n(t) = \sum_{\mathcal{J}_{(i,k,x)}:x=n} dbf_{(i,k)}(t)$$

$dbf_n(t)$ upper bounds the largest resource demand of all the subtasks in node N_n over any time interval of length t. Liu and Layland [18] have shown that $d_{(i,k)} \leq D_{(i,k)}$ holds for each subtask $\mathcal{J}_{(i,k,n)}$ if

$$dbf_n(t) \leq t \times \mathsf{UB}_n, \forall t > 0, \tag{4}$$

where UB_n represents the utilization bound of the scheduling algorithm used by node N_n. It is defined as

$$\mathsf{UB}_n = \begin{cases} 1 & \text{under preemptive EDF [18]} \\ 1 - \delta_n & \text{under non-preemptive EDF [9]} \end{cases} \tag{5}$$

where δ_n is $\max_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}}$. For convergence of the optimization technique that we develop in Section 3, all the constraints used in the optimization problem must be concave functions of the variables. In that framework, local subtask deadlines $D_{(i,k)}$ are the variables, and the schedulability conditions that we develop here are used as constraints. Hence Eq. (4) cannot be directly used in that framework, because $dbf_n(t)$ is not a concave function of the local deadlines. Therefore, we now present an upper bound for $dbf_n(t)$ such that the corresponding schedulability conditions can be used in our optimization framework. Consider the following density function.

$$den_{(i,k)}(t) = \frac{C_{(i,k)}}{D_{(i,k)}} \cdot t$$

It is easy to see that $den_{(i,k)}$ is a tight, linear, upper bound of $dbf_{(i,k)}$, *i.e.*, $dbf_{(i,k)}(t) \leq den_{(i,k)}(t)$ for all $t > 0$, and they are equal when $t = D_{(i,k)}$.

Using this upper bound in Eq. (4), we get the following conditions for schedulability of subtasks in N_n.

$$\sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} \leq \mathsf{UB}_n, \forall t > 0 \tag{6}$$

For the schedulability constraint given above, the following statement is true for all $D_{(i_0,k_0)} > 0$.

$$\frac{\partial^2}{\partial D^2_{(i_0,k_0)}} \left[\mathsf{UB}_n - \sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} \right] \leq 0$$

Hence the constraint is a concave function of deadline variables and therefore it can be used in the optimization framework that we develop in Section 3.

2.4 Failure model

Failure models have been studied to tolerate faults [25, 26, 23]. We consider a failure model assuming that any transient failures can occur during executions and detection of the failures is possible within the maximum time[2]. Once a subtask experiences transient failure such as system error or packet loss during execution, subtasks can re-execute after failure detection, and they are still subject to additional failures during these re-executions. Let $P_{(i,k)}$ denote the probability that failure occurs to a single execution of a job of $\mathcal{J}_{(i,k)}$. Since a single job of $\mathcal{J}_{(i,k)}$ can experience one or more failures in its lifetime, let $F_{(i,k)}$ denote the number of such failures. Finally, let $\mathbf{P}\{F_{(i,k)} = m\}$ represent the probability that a job of $\mathcal{J}_{(i,k)}$ experiences exactly m failures and then finally one success. If failures occur independently of each other, we get:

$$\mathbf{P}\{F_{(i,k)} = m\} = (1 - P_{(i,k)}) \cdot (P_{(i,k)})^m. \tag{7}$$

Note that the above failure model (independent failures) is just one example, and as long as $\mathbf{P}\{F_{(i,k)} = m\}$s are given, our framework can be applied. We assume that the worst-case execution time of re-execution of $\mathcal{J}_{(i,k)}$ is also $C_{(i,k)}$. This assumption is for better explanation of deriving equations in later sections, and our framework can be easily extended to the model with larger (due to time to detect failure) or smaller (due to efficient failure recovery procedure) worst-case re-execution time.

Our failure model enables to transform a complex problem dealing with random failures to a more manageable problem where the randomness is bounded through probabilistic execution.

[2]Fault detection techniques have been widely studied [23, 25]. An example of techniques to justify the assumption is to use timeout of acknowledgement packets.

2.5 An example of the system model

There are many systems [3, 30, 17] that can be modeled with our system model, and we exemplify our system model by a system called office enterprise security and hazard sensing environment [3]. Sensor nodes monitor circumference such as movement of any objects, temperature, and humidity, and they send the raw data to office enterprise network. Nodes in the network process the data (*e.g.*, filter and analyze the data, update database [17]), and they relay the processed data to the destination nodes for security monitoring or video conferencing. Here we can regard transmission of the data through network as network scheduling and processing of the data as CPU scheduling. QoS for these time-critical tasks can be modeled using end-to-end delay-sensitive utility functions in that a better QoS is implied by more prompt response in sensor monitoring or more interactive communication in video conferencing. The system needs online scheduling to be adaptive to dynamically changing environments (*e.g.*, joining/leaving of participants for video conferencing, turning on/off sensor nodes to save power in security monitoring) and unpredictable errors (*e.g.*, dynamic system failures, packet loss).

3. OPTIMIZATION FRAMEWORK

3.1 Deadline assignment problem

The *deadline assignment* problem aims to determine the local deadline ($D_{(i,k)}$) of every subtask ($\mathcal{J}_{(i,k)}$) in order to provide a guaranteed maximum system utility (maximum U_{sys}). Thus the problem can be formulated as

(Primal problem)

$$\text{Max: } U_{sys} = \sum_{\tau_i \in \mathcal{T}_{sys}} U_i \left(\sum_{k=1}^{m_i} D_{(i,k)} \right), \tag{8}$$

$$\text{Sub. to: } \sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} \leq \mathsf{UB}_n, \forall n : n \in 1, \ldots, V_N \tag{9}$$

In order to provide guarantees over end-to-end delays and thereby a resulting system utility, we require that the node schedulability constraint of Eq. (9) be valid for all nodes. Otherwise, the utility may exhibit unstable behavior because of larger than expected subtask delays. Therefore a solution to the above optimization problem is said to be *valid* if Eq. (9) holds for all nodes.

Eq. (9) is a set of concave constraints on local deadline variables. Since each U_i is a concave and decreasing function (discussed in Section 2), the summation in Eq. (8) is also concave. Therefore the above primal problem is naturally a concave optimization problem.

3.2 Distributed optimization framework

In many practical scenarios, distributed optimization schemes offer more advantages over centralized ones. Distributed computations, which make decisions based on local information, are more scalable, efficient and robust when compared to centralized methods that rely on global information. Therefore, in this subsection, we present a distributed computation technique for the above optimization problem.

Any optimization problem can be re-written in its dual form using Lagrange multipliers (see Chapter 5 in [8]). This formulation is called the *Lagrange dual problem*. For the formulation presented in the previous subsection, called "primal" formulation, its Lagrange dual is given as

(Dual problem)

$$\min_{\mathbf{p}\geq 0} \max_{\mathbf{D}\geq 0} L(\mathbf{D},\mathbf{p}) = \sum_{\tau_i \in \mathcal{T}_{sys}} U_i \left(\sum_{k=1}^{m_i} D_{(i,k)} \right) + \sum_{n:n\in 1,\ldots,V_N} p_n \cdot \left(\mathsf{UB}_n - \sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} \right) \quad (10)$$

$$\text{Sub. to: } p_n \geq 0, \forall n \in 1,\ldots,V_N \quad (11)$$

where $\mathbf{D} = \{D_{(i,k)}\}, \forall (i,k) \in \{(s,w) | \tau_s \in \mathcal{T}_{sys}, w = 1,\ldots,m_s\}$ and $\mathbf{p} = \{p_n\}, \forall n \in 1,\ldots,V_N$. In the above problem formulation, each p_n is a Lagrange multiplier that denotes the price for satisfying the schedulability constraint of node N_n. In optimization theory, the *duality gap* of a problem represents the difference between the optimal solutions of its primal and dual formulations. This gap is zero when the primal problem is a concave optimization problem (see Chapter 5 in [8]), and this means that an optimal solution to the dual problem is equivalent to an optimal solution to the primal problem. Therefore, we can obtain an optimal solution to the deadline assignment problem by finding optimal solutions to the dual problem.

We can find node prices p_n that collectively minimize $L(\mathbf{D},\mathbf{p})$ through gradient method [6, 19] as follows.

$$p_n(t+1) = \left[p_n(t) - \gamma_n \cdot \left(\mathsf{UB}_n - \sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} \right) \right]^+ \quad (12)$$

where $p_n(t)$ means the value of p_n at the t^{th} iteration and $[x]^+$ means $\max\{0,x\}$. The constant γ_n controls the step-size between iterations and also affects the rate of convergence. If it satisfies Lipschitz continuity [6], then the iterations are guaranteed to converge. We can also obtain local deadlines $D_{(i,k)}$ that maximize $L(\mathbf{D},\mathbf{p})$ by solving the following differential equation:

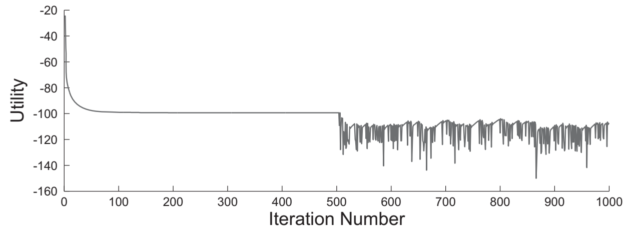
$$\frac{\partial L(\mathbf{D},\mathbf{p})}{\partial D_{(i,k)}(t+1)} = 0 \quad (13)$$

We calculate the above equation in conjunction with Eq. (12), and then obtain the local deadline $D_{(i,k)}$ in the $(t+1)^{th}$ iteration (denoted as $D_{(i,k)}(t+1)$). In this iteration, the other deadlines $D_{(i,x)}$ are regarded as constants (where $x \neq k$), and their values from the t^{th} iteration are used ($D_{(i,x)}(t)$).

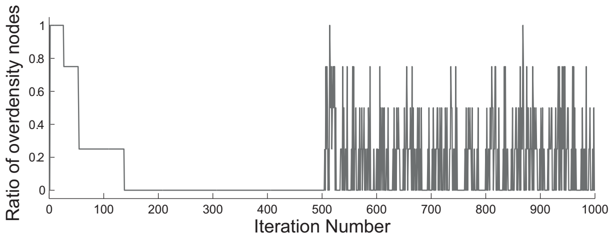
4. ROBUST OPTIMIZATION FRAMEWORK

In the previous section we presented a distributed QoS optimization framework for the deadline assignment problem (henceforth denoted as the "nominal optimization framework"). The presented framework makes it possible for all local deadlines to converge to a global optimal solution in a distributed manner. An important premise employed in the nominal formulation is that every subtask is free from failure. However, subtasks are subject to unpredictable failures in many practical environments. Typical failures include execution failures from transient hardware faults and communication failures from network errors.

In this paper, we assume that each subtask $\mathcal{J}_{(i,k)}$ is subject to unpredictable failures and $P_{(i,k)}$ represents the probability that a



(a) Utility



(b) Ratio of over-density nodes

Figure 3: Behavior of nominal optimization framework under failure

single failure happens to a single job of $\mathcal{J}_{(i,k)}$. We assume that the job executes again to recover from this failure, imposing an extra execution time of $C_{(i,k)}$. As failures occur randomly, re-executing jobs can repeatedly suffer from failures. This adds additional execution times to jobs in an unpredictable manner and embeds uncertainty into the worst-case execution time of subtasks. Such uncertainty potentially leads the nominal optimization framework to exhibit poor behavior. For instance, the QoS guaranteed by the nominal framework may not be achievable as subtasks experience random failures. As an example, we consider a simple task set as shown in Figure 2(b). A subtask $\mathcal{J}_{(i,k)}$ has $C_{(i,k)} = 1$ for $i = 1$, and $C_{(i,k)} = 2$ for $i \geq 2$. Suppose each subtask $\mathcal{J}_{(i,k,4)}$ starts experiencing random failures with a failure probability $P_{(i,k,4)} = 0.1$ from the 500^{th} iteration. Figure 3(a) illustrates such an unstable behavior; the system utility keeps fluctuating as failures occur, and the guaranteed QoS of -100 is not met[3]. This happens because the uncertainty in worst-case execution time of subtasks invalidates some of the node schedulability constraints of Eq. (9), and as a result local subtask deadlines are not met. Figure 3(b) plots the ratio of nodes with density greater than UB_n to the total number of nodes. As shown, this ratio increases in an unpredictable manner with node failures.

Handling of unpredictable failures entails an optimization framework that can produce a "robust" (valid) solution under perturbations of subtask worst-case execution times. Many issues need to be addressed in order to make our nominal optimization framework robust: 1) we must understand how random failures introduce uncertainty into the worst-case execution time of subtasks, 2) we must characterize this uncertainty in order to accommodate random perturbations in node-level schedulability, taking into consideration the conditions imposed by Lagrangian duality theory, and 3) we must define a proper notion of robustness to cope with the uncertainty effectively.

[3]In Figure 3(a) and (b) seem to illustrate unstable behaviors at the beginning because each deadline has not yet converged. Once each deadline converges to the optimal point, there is no more unstable behavior in case of no failure.

4.1 Uncertainty set

When a subtask fails to execute, it re-executes to recover from the failure. This imposes an extra execution time and subsequently increases the node density. This increment potentially leads to invalidation of the node schedulability constraint of Eq. (9), thereby invalidating the QoS guarantee. We require the node schedulability constraints to be valid under some failures in order to preserve the QoS guarantee of the optimization framework.

Consider a subtask $\mathcal{J}_{(s,t)}$ on a node N_n. Suppose it experiences a failure during execution, and it demands another execution of $C_{(s,t)}$ prior to its deadline. This additional execution requirement can be successfully satisfied if some spare time no smaller than $C_{(s,t)}$ has been reserved in advance. Such a reservation can be viewed as adding the density for another (new) subtask into the schedulability constraint. For instance, we can successfully save spare time enough to re-execute $C_{(s,t)}$ by its deadline if the following condition holds:

$$\sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} + \frac{C_{(s,t,n)}}{D_{(s,t,n)}} \leq \mathsf{UB}_n \tag{14}$$

Failures happen to subtasks in an unpredictable manner. Subtasks can experience even additional failures while they are executing again to recover from earlier failures. Therefore subtasks can experience a different number of failures randomly. This inherently embeds uncertainty into the extra execution times reserved for failure recovery. Recall that $F_{(i,k)}$ denotes the maximum number of failures that can be experienced by a single job of subtask $\mathcal{J}_{(i,k)}$. We now show how to characterize the uncertainty in failure-related (extra) execution times using $F_{(i,k)}$. As $F_{(i,k)}$ for each subtask in a node can be different, there are a number of possible combinations of $F_{(i,k)}$'s on each node. We refer to a single instance of the combinations within a node N_n as an "uncertainty instance" X_n. More formally,

$$X_n = (F_{(i_1,k_1)}, F_{(i_2,k_2)}, ...), \tag{15}$$

where $(i_l, k_l) \in \{(s,t) | \mathcal{J}_{(s,t,x)} : x = n\}$. Let $X_n.l$ denote the l-th element of X_n. X_n indicates that every subtask $\mathcal{J}_{(i,k)}$ experiences at most $F_{(i,k)}$ failures, where $F_{(i,k)} \in X_n$. Using this uncertainty instance, we can define an entire uncertainty set (denoted as $\mathcal{A}_n$) for a node N_n, which is a set of all possible combinations of $F_{(i,k)}$, *i.e.*, all possible X_n. We note that the number of elements in $\mathcal{A}_n$ is theoretically infinite.

We can then generalize Eq. (14) in order to provide $\mathcal{J}_{(i,k)}$ with some spare execution time, so that it can recover from at most $F_{(i,k)}$ failures. This generalization is given by

$$\sum_{\mathcal{J}_{(i,k,x)}:x=n} (1 + F_{(i,k)}) \cdot \frac{C_{(i,k)}}{D_{(i,k)}} \leq \mathsf{UB}_n \tag{16}$$

4.2 Uncertainty set with K-failure

We wish to derive a notion of robustness coping with uncertainty. The uncertainty set $\mathcal{A}_n$ is generally too large, and hence it is hard to develop a proper notion of robustness based on it. Therefore, we first categorize its elements to find some useful property per category in this subsection. Then, in the next subsection, we derive a notion of robustness from this property (called K-robustness: being robust to at most K concurrent failures), and show how K-robustness is incorporated with the robust formulation.

We classify the uncertainty set based on the notion of concurrent failures. Let $a_{(i,k)}$ denote the release time of a subtask $\mathcal{J}_{(i,k)}$. We define a set of subtasks to be *concurrently running* if there exists some time t such that $a_{(i,k)} \leq t \leq a_{(i,k)} + D_{(i,k)}$ for each subtask $\mathcal{J}_{(i,k)}$ in the set. Failures are then defined to be *concurrent*, if they occur together for a set of concurrently running subtasks. For a given uncertainty instance X_n, the number of concurrent failures is maximized when all concurrently running subtasks $\mathcal{J}_{(i,k)}$ experience $F_{(i,k)}$ failures together, where $F_{(i,k)} \in X_n$. The maximum number of concurrent failures for X_n is then calculated as

Uncertainty instances with K-failure $X_n = (F_{(1,1)}, F_{(2,1)})$	K	$\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\}$ with $P_{i,k} = 0.1$	$\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\}$ with $P_{i,k} = 0.01$
(0,0)	0	0.8100	0.9801
(0,0), (1,0), (0,1)	1	0.9720	0.9997
(0,0), (1,0), (0,1) (2,0), (1,1), (0,2)	2	0.9963	0.9999
(0,0), (1,0), (0,1), (2,0), (1,1) (0,2), (3,0), (2,1), (1,2), (0,3)	3	0.9995	0.9999
⋮	⋮	⋮	⋮

Table 1: Uncertainty instances with K-failure

$$M(X_n) = \sum_{l=1}^{|X_n|} X_n.l \tag{17}$$

where $|X_n|$ is the number of elements in tuple X_n. We define an uncertainty instance X_n to be associated with K-*failure*, if X_n can contain at most K concurrent failures (i.e., $M(X_n) \leq \mathsf{K}$). As an example, we consider two subtasks $\mathcal{J}_{(1,1,n)}$ and $\mathcal{J}_{(2,1,n)}$ in N_n. Table 1 shows some uncertainty instances with K-failure. The first column of the table shows uncertainty instances $X_n = (F_{(1,1)}, F_{(2,1)})$, and the second column represents the maximum number of concurrent failures (denoted as K).

We now consider the probability (denoted by $\mathbf{P}\{X_n\}$) that a single uncertainty instance X_n can occur, when each subtask $\mathcal{J}_{(i,k)}$ experiences a failure independently[4] with its own failure probability $P_{(i,k)}$.

$$\mathbf{P}\{X_n\} = \prod_{l=1}^{|X_n|} \mathbf{P}\{F_{(i_l,k_l)} = X_n.l\}, \tag{18}$$

where $\mathbf{P}\{F_{(i_l,k_l)} = X_n.l\}$ is given by Eq. (7). Now we define $\mathcal{A}_n(\mathsf{K})$ to denote a set of uncertainty instances with K-failure, i.e.,

$$\mathcal{A}_n(\mathsf{K}) = \{X_n | M(X_n) \leq \mathsf{K}, X_n \in \mathcal{A}_n\} \tag{19}$$

Then we can define the probability (denoted by $\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\}$) that an uncertainty set with K-failure can occur, as the sum of probability of its individual elements:

$$\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\} = \sum_{X_n \in \mathcal{A}_n(\mathsf{K})} \mathbf{P}\{X_n\}. \tag{20}$$

Table 1 also shows the probability of uncertainty set with K-failure in the third column when each subtask $\mathcal{J}_{i,k}$ has $P_{i,k} = 0.1$, and in the fourth column when each subtask $\mathcal{J}_{i,k}$ has $P_{i,k} = 0.01$, respectively. As an example, $\mathbf{P}\{\mathcal{A}_n(\mathsf{K}=0)\}$ and $\mathbf{P}\{\mathcal{A}_n(\mathsf{K}=1)\}$ when $P_{i,k} = 0.1$ are computed as follows: each subtask has a probability of having zero failure as 0.9 $(= 1.0 - 0.1)$, and they

[4]For simplicity, we assume independence of failures, but if they are correlated, then we can easily apply the correlation in our framework.

have both zero failure ($X_n = (0,0)$) at the probability of 0.81 ($= 0.9 \cdot 0.9$) so that $\mathbf{P}\{\mathcal{A}_n(\mathsf{K}=0)\}$ is equal to 0.81. Likewise, the probability that each task experiences exactly one failure and then a success, is $0.9 \cdot 0.1 = 0.09$. Then the probability ($X_n = (1,0)$) is 0.081 ($= 0.09 \cdot 0.9$) that a subtask experiences one failure and then a success while another subtask experiences a success without any failure. Since $\mathbf{P}\{\mathcal{A}_n(\mathsf{K}=1)\}$ includes three uncertain instances (0,0), (1,0), and (0,1), it equals to 0.972 ($= 0.9 \cdot 0.9 + 0.09 \cdot 0.9 + 0.9 \cdot 0.09$).

4.3 Robust formulation with K-robustness

In this subsection, we introduce a notion of robustness on the basis of the notion of K-failure. A node N_n is said to be robust to K-failure (or K-*robust*), if it can keep all of its subtasks schedulable under any uncertainty set with K-failure. The following theorem presents a schedulability condition for K-robust nodes.

THEOREM 1. *A node N_n is* K*-robust, if*

$$\sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} + \mathsf{K} \cdot \max_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} \le \mathsf{UB}_n. \quad (21)$$

PROOF. Let us assume that we have a set of subtasks for which Eq. (21) holds. We consider an uncertainty set with K-failure, that is each $\mathcal{J}_{(i,k)}$ experiences $F_{(i,k)}$ failures and $\sum_{\mathcal{J}_{(i,k,x)}:x=n} F_{(i,k)} \le \mathsf{K}$. Using this bound condition, we can see that the density of the uncertainty set (LHS of Eq. (16)) is less than or equal to the LHS of Eq. (21).

Since the schedulability condition holds, node N_n can keep all of its subtasks schedulable under this uncertainty set with K-failure. Since we made no assumptions about the uncertainty set (except that it has K-failure), we get that the above statement holds for any uncertainty set with K-failure. Therefore node N_n is K-robust. □

The above theorem, although somewhat obvious, is important because it enables us to incorporate random failures into the optimization problem, without violating the Lagrange requirements for distributed computation.

$\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\}$ in Eq. (20) can be interpreted as the probability that a K-robust node N_n produces valid solutions under random failures. Let us define the protection function of a node N_n, $\mathbf{G}_n(\mathsf{K})$, as a function that represents the difference between the nominal schedulability constraint of Eq. (9) and the K-robust schedulability constraint of Eq. (21). That is,

$$\mathbf{G}_n(\mathsf{K}) = \mathsf{K} \cdot \max_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}}. \quad (22)$$

Thus, our nominal optimization framework for the deadline assignment problem can be made robust against K-failure as follows.

(Robust formulation)

$$\text{Max: } U_{sys} = \sum_{\tau_i \in \mathcal{T}_{sys}} U_i \left(\sum_{k=1}^{m_i} D_{(i,k)} \right), \quad (23)$$

$$\text{Sub. to: } \sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} + \mathbf{G}_n(\mathsf{K}) \le \mathsf{UB}_n,$$
$$\forall n : n \in 1, \ldots, V_N. \quad (24)$$

In this robust formulation, K controls the trade-off between robustness and performance with a probability. Such a trade-off will be explored in Section 6. Further, Equation (24) remains valid even when the actual execution time of subtask $J_{(i,k)}$ is smaller than $C_{(i,k)}$ or the actual number of subtask failures in a node is smaller than K. Therefore the above formulation is also sustainable with respect to the execution time of subtasks. Finally, the formulation is a standard convex optimization problem, because the only modification done to the nominal framework is addition of a convex and decreasing protection function ($\mathbf{G}_n(\mathsf{K})$) to the schedulability constraints. Therefore the duality gap of this robust formulation is zero, and we can compute the optimal solution using its dual formulation. The Lagrange dual problem of this robust formulation is given as follows:

(Dual problem of robust formulation)

$$\min_{\mathbf{p}\ge 0} \max_{\mathbf{D}\ge 0} L(\mathbf{D},\mathbf{p}) = \sum_{\tau_i \in \mathcal{T}_{sys}} U_i \left(\sum_{k=1}^{m_i} D_{(i,k)} \right)$$
$$+ \sum_{n:n\in 1,\ldots,V_N} p_n \cdot \left(\mathsf{UB}_n - \sum_{\mathcal{J}_{(i,k,x)}:x=n} \frac{C_{(i,k)}}{D_{(i,k)}} - \mathbf{G}_n(\mathsf{K}) \right)$$
$$\text{Sub. to: } p_n \ge 0, \forall n \in 1, \ldots, V_N$$

We can find the optimal solution by solving the dual problem, just as described in Section 3.2.

5. DISTRIBUTED COMPUTATION

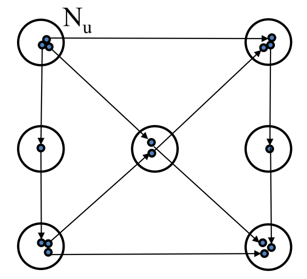
In the previous sections we presented optimization frameworks that perform distributed computations to collectively evolve to an optimal solution. This section discusses some implementation issues. They include 1) what kind of control messages should be exchanged, 2) what will happen if some control messages are lost, and 3) how to define a convergence criteria for our optimization frameworks.

5.1 Control messages

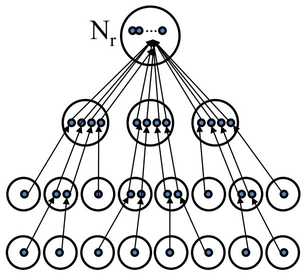
The Lagrangian dual formulations identify what information needs to be exchanged in order to carry out distributed optimization. Specifically, Eq. (12) shows how to compute a node price p_n at each node N_n, and it requires knowledge of all the subtask deadlines in the same node N_n. Eq. (13) shows how to compute a subtask deadline $D_{(i,k,n)}$, and it requires knowledge of p_n and the local deadlines of τ_i's other subtasks. That is, solving Eq. (13) requires cross-node communication. We refer to the information (i.e., p_n and $D_{(i,k)}$) to be exchanged as "control message". Exchange of control messages can be effectively implemented with little extra communication cost. For example, many approaches to the network utility maximization problem employ efficient mechanisms to exchange implicit information (*e.g.*, congestion price marked in packets, loss rate, or some piggybacked values) with no extra packet delivery [1].

5.2 Loss of control messages

For many practical environments, exchange of control messages can also fail, and one may wonder the effect of such failures on our distributed computations. Fortunately, our optimization frameworks can converge to an optimal solution, even in the presence of such control message losses. For example, when a control message is lost at some iteration step, the frameworks can use the control message from the previous step. This asynchronous iteration reduces the rate of convergence, but still guarantees convergence [6].



(a) A general distributed system



(b) A sensor network

Figure 4: Network topology in simulation

A key idea of the proof [6] is to set a worst-case period by which the control messages become outdated, and the rest of the proof is similar to the case of synchronous iterations. Section 6 will illustrate the effect of loss of control messages on convergence.

5.3 Convergence criteria

Another implementation issue is how to determine when the iterative computation of Eqs. (12) and (13) converge. We define our convergence criteria:

$$|p_n(t+1) - p_n(t)| < \epsilon_p \tag{25}$$

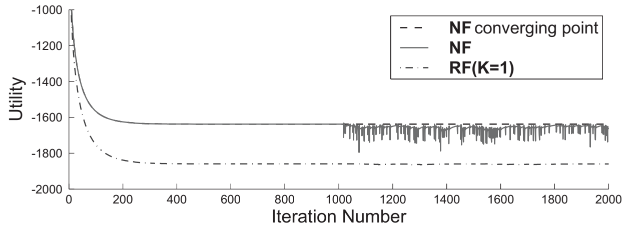
$$|D_{(i,k)}(t+1) - D_{(i,k)}(t)| < \epsilon_D \tag{26}$$

where ϵ_p and ϵ_D are sufficiently small positive real numbers; they generate a trade-off between accuracy and rate of convergence. Many gradient algorithms generally employ this kind of convergence criteria [6].

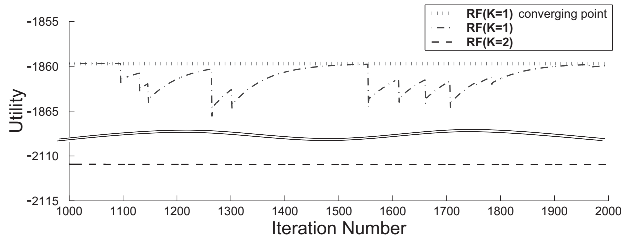
6. PERFORMANCE EVALUATION

This section presents (MATLAB-based) simulation results to illustrate the performance of the proposed optimization frameworks in two aspects. We first evaluate the behavior of our robust framework under failure in comparison to the nominal framework. We then illustrate the effect of control message loss on our distributed optimization framework.

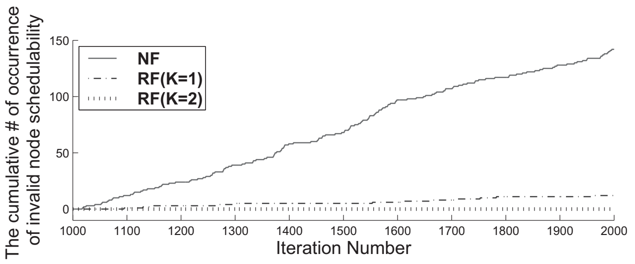
We consider two simulation settings with different network topologies. The first simulation setting includes a network topology that represents a general distributed real-time system, as shown in Figure 4(a). It contains six tasks over seven nodes, and each task consists of multiple subtasks across nodes, as shown in the figure. The second simulation setting involves another topology, shown in Figure 4(b), that represents a typical sensor network that collects sensor data at leaf nodes and relays the data through intermediate nodes to the root node. It has 20 nodes forming a tree structure, and 12 tasks execute from leaf nodes to the root node, as shown in the figure. The worst-case execution times ($C_{(i,k)}$) of subtasks are randomly chosen from the interval $[1, 5]$ for the first setting and from the interval $[1, 3]$ for the second setting. For both the simulation settings, we use utility functions $U_i(x) = -\frac{1}{2}x^2$ for each task τ_i, unless specified otherwise. For convenient reference, let NF represent our nominal distributed optimization framework, and RF(K) represent our robust distributed optimization framework with K-robustness.



(a) Utility



(b) Utility



(c) The cumulative # of occurrence of invalid node schedulability

Figure 5: Robust frameworks over the first simulation setting

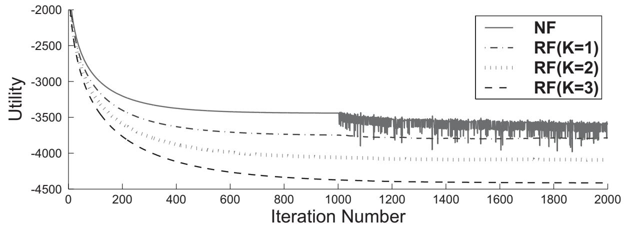
6.1 Robust optimization framework

In order to show the effect of random failures on nominal and robust optimization frameworks, we run simulations on both the simulation settings for 2000 steps such that the first half involves no failure and the second half contains random failures. Random failures are added into the 3 subtasks executing in the upper left node in Figure 4(a) (denoted as N_u) and the 12 subtasks executing in the root node in Figure 4(b) (denoted as N_r). Each of these subtasks experiences a failure with a probability of 5%.

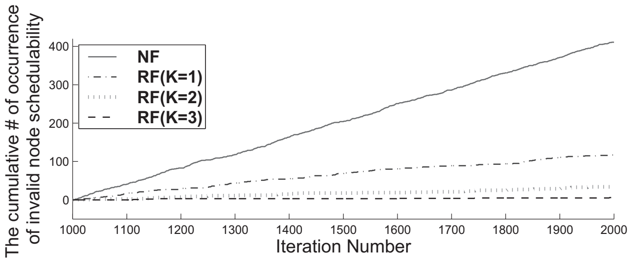
Figure 5(a) shows the behavior of NF and RF over the first simulation setting. It is shown in the figure that under NF, the system utility converges during the first 1000 steps, but it becomes unstable as random failures occur from the 1000th step onwards. On the other hand, the figure shows that under RF(K=1), the system utility remains stable in a large scale, however it is small-scale fluctuating as failures occur (see zoom in Figure 5(b)). In fact, RF(K=1) is fully robust when the number of maximum possible concurrent failures is bounded by one (i.e., robust to the failures captured by $\mathcal{A}_u$(K=1)). However, as simulations may contain two or more concurrent failures[5], RF(K=1) shows small-scale fluctuations over such failures. Figure 5(b) also shows that under a more robust framework, such as RF(K=2), it is able to remain stable without any fluctuations over the same failures.

Figure 5(c) shows another aspect of our optimization frameworks.

[5]Since failures may occur independently of each other and may consecutively occur due to the failure of re-executions, the number of concurrent failures can be any non-negative integer value.



(a) Utility



(b) The cumulative # of occurrence of invalid node schedulability

Figure 6: Robust frameworks over the second simulation setting

Failures often invalidate node-level schedulability and consequently make resulting solutions invalid. Figure 5(c) plots the number of occurrences of invalid node-level schedulability constraint at node N_u under NF, RF(K=1) and RF(K=2), as iterations continue. The figure shows that as failures continue to occur, such invalid schedulability constraints keep increasing rapidly under NF, increases very slowly under RF(K=1), and remains zero under RF(K=2), indicating that a more robust framework also produces more valid solutions. This is because if the number of current failures is more than K, end-to-end deadline may not be met and then guaranteed utility may not be achieved.

Figure 6 shows simulation results over the second simulation setting. It basically shows that the simulation results obtained from the first simulation setting are also applicable in the second setting. Figure 6(a) shows that under NF, the system utility becomes noticeably unstable under failures, but under robust frameworks, it remains stable in a large scale. It is still marginally fluctuating under robust frameworks, similarly to the behavior shown in Figure 5(b). Figure 6(b) shows that the occurrences of invalid node-level schedulability constraints decrease sharply as the framework becomes more robust (i.e., K increases). The actual numbers of such occurrences are shown in Table II with their expected values. Table II also shows that as K increases, the robustness probability ($\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\}$) increases, but the system utility decreases[6]. Here we can see that K serves as a parameter to control the trade-off between QoS guarantees and probabilistic robustness.

6.2 Loss of control messages

Our optimization framework requires exchange of some control messages ($D_{(i,k)}$ and p_n) to perform distributed computations. Here we evaluate the performance of the framework in the presence of loss of some control messages. We use the first simulation setting. We deliberately drop some control messages with some loss probability (0.0, 0.2, 0.4, 0.6, or 0.8). When a control message is lost, the framework uses the previous message to recover the loss. Figure 7 shows that the rate of convergence reduces as the loss probability increases. It also shows that the framework converges even with a loss probability of 80%, at the expense of convergence rate. These results go with the explanation in Section 5.2. We can see that loss of control messages only affects convergence rate, not convergence itself.

[6]In table II, the expected value is different from the actual cumulative number of occurrence because the actual number is just from a sample path of experiment.

K	$\mathbf{P}\{\mathcal{A}_n(\mathsf{K})\}$	cumulative # of occurrence of invalid node-level schedulability (expected value)	Utility
0	0.5404	390 (460)	-3443
1	0.8646	117 (135)	-3754
2	0.9700	34 (30)	-4078
3	0.9946	6 (5)	-4412

Table 2: Robustness and achievable QoS with K

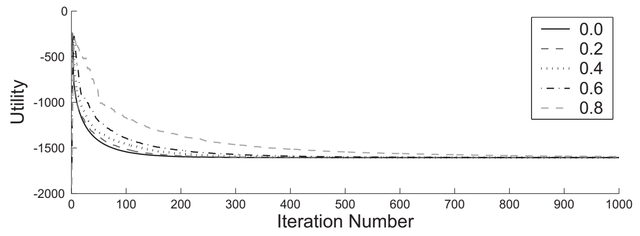


Figure 7: Convergence behavior under loss of control messages

7. CONCLUSION

This paper presented a robust distributed optimization framework maximizing the aggregate QoS in distributed soft real-time systems, particularly, to effectively address the deadline assignment problem in the presence of unpredictable failures. It offers a solid foundation that translates uncertainty (due to random failures) to probabilistically robust schedulability.

Several aspects of the framework are directions for further research. Our framework mainly accepts convex and concave constraints. However, most efficient schedulability conditions [5, 2] do not satisfy this property. Hence one direction is to develop a new tight, convex or concave schedulability condition. Our notions of K-failure and K-robustness are tightly coupled with node-level schedulability. Another interesting direction is to extend such notions towards the entire system schedulability. This raises challenges of developing new analysis techniques for distributed real-time systems, which makes it possible to analyze the schedulability of an entire distributed system in a holistic manner.

8. ACKNOWLEDGEMENT

This work was supported in part by the IT R&D Program of MKE/KEIT [2010-KI002090, Development of Technology Base for Trustworthy Computing], Basic Research Laboratory (BRL) Program (2009-0086964) and Basic Science Research Program (2010-0006650) through the National Research Foundation of Korea(NRF) funded by the Korea Government(MEST), and KAIST ICC, KI-DCS, and KMCC grants.

This work was also partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the European Commission through grant ArtistDesign ICT-NoE-214373.

9. REFERENCES

[1] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: Active Queue Management. *IEEE Network*, pages 48–53, May 2001.

[2] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheudling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] J. Balasubramanian, S. Tambe, B. Dasarathy, S. Gadgi, F. Porter, A. Gokhale, and D. C. Schmidt. NetQoPE: A model-driven network QoS provisioning engine for distributed real-time and embedded systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 113–122, 2008.

[4] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 159–168, 2006.

[5] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time Systems*, 2:401–424, 1990.

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[7] E. Bini and A. Cervin. Delay-aware period assignment in control systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 291–300, 2008.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[9] G. C. Butazzo. *Hard Real-Time Computing Systems*. Springer, 2005.

[10] Y. Chen, C. Lu, and X. Koutsoukos. Optimal discrete rate adaptation for distributed real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 181–192, 2007.

[11] R. I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 3–14, 2007.

[12] J. J. G. Garcia and M. G. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems*, pages 124–132, 1995.

[13] P. Jayachandran and T. Abdelzaher. Bandwidth allocation for elastic real-time flows in multihop wireless networks based on network utility maximization. In *Proceedings of International Conference on Distributed Computing Systems*, pages 752–759, 2008.

[14] P. Jayachandran and T. Abdelzaher. Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 259–269, 2008.

[15] P. Jayachandran and T. Abdelzaher. End-to-end delay analysis of distributed systems with cycles in the task graph. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 13–22, 2009.

[16] J. Jonsson and K. G. Shin. Deadline assignment in distributed hard real-time systems with relaxed locality constraints. In *Proceedings of International Conference on Distributed Computing Systems*, pages 432–440, 1997.

[17] A. Kavimandan and A. Gokhale. Automated middleware QoS configuration techniques for distributed real-time and embedded systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 93–102, 2008.

[18] C. Liu and J. Layland. Scheduling algorithms for multi-programming in a hard-real-time environment. *Journal of the ACM*, 20(1):46 – 61, 1973.

[19] S. H. Low. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, pages 861–874, Dec. 1999.

[20] C. Lumezanu, S. Bhola, and M. Astley. Online optimization for latency assignment in distributed real-time systems. In *Proceedings of International Conference on Distributed Computing Systems*, pages 752–759, 2008.

[21] 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*, pages 216–227, 1994.

[22] J. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 3–12, 2003.

[23] L. Paradis and Q. Han. A survey of fault management in wireless sensor networks. *Journal of Network and Systems Management*, 15:171–190, 2007.

[24] W. Shu, X. Liu, Z. Gu, and S. Gopalakrishnan. Optimal Sampling Rate Assignment with Dynamic Route Selection for Real-Time Wireless Sensor Networks. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 431–441, 2008.

[25] A. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms (2nd ed.)*. Prentice Hall, 2007.

[26] M. Treaster. A survey of fault-tolerance and fault-receovery techniques in parallel systems. *ACM Computing Research Repository*, abs/cs/0501002, 2005.

[27] X. Wang, X. Fu, X. Liu, and Z. Gu. Power-aware CPU utilization control for distributed real-time systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 233–242, 2009.

[28] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. Time/utility function decomposition techniques for utility accrual scheduling algorithms in real-time distributed systems. *IEEE Transactions on Computers*, 54(9):1138–1153, 2005.

[29] K. Yang, Y. Wu, J. Huang, X. Wang, and S. Verdu. Distributed Robust Optimization for Communication Networks. In *Proceedings of the 27th Conference on Computer Communications*, pages 1157–1165, 2008.

[30] Y. Zhao, J. Liu, and E. A. Lee. A programming model for time-synchronized distributed real-time systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 259–268, 2007.

[31] Q. Zhu, Y. Yang, E. Scholte, M. D. Natale, and A. S.-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 275–284, 2009.