Report Number: WUCSE-2014-48

2014

Federated Scheduling for Stochastic Parallel Real-time Tasks

Jing Li, Kunal Agrawal, Christopher Gill, and Chenyang Lu

Federated scheduling is a strategy to schedule parallel real-time tasks: It allocates a dedicated cluster of cores to high-utilization task (utilization >1); It uses a multiprocessor scheduling algorithm to schedule and execute all low-utilization tasks sequentially, on a shared cluster of the remaining cores. Prior work has shown that federated scheduling has the best known capacity augmentation bound of 2 for parallel tasks with implicit deadlines. In this paper, we explore the soft real-time performance of federated scheduling and address the average-case workloads instead of the worst-case values. In particular, we consider stochastic tasks -- tasks for which execution... **Read complete abstract on page 2.**

Federated Scheduling for Stochastic Parallel Real-time Tasks

Jing Li, Kunal Agrawal, Christopher Gill, and Chenyang Lu

Complete Abstract:

Federated scheduling is a strategy to schedule parallel real-time tasks: It allocates a dedicated cluster of cores to high-utilization task (utilization >1); It uses a multiprocessor scheduling algorithm to schedule and execute all low-utilization tasks sequentially, on a shared cluster of the remaining cores. Prior work has shown that federated scheduling has the best known capacity augmentation bound of 2 for parallel tasks with implicit deadlines. In this paper, we explore the soft real-time performance of federated scheduling and address the average-case workloads instead of the worst-case values. In particular, we consider stochastic tasks -- tasks for which execution time and critical-path length are random variables. In this case, we use bounded expected tardiness as the schedulability criterion. We define a stochastic capacity augmentation bound and prove that federated scheduling algorithms guarantee the same bound of 2 for stochastic tasks. We present three federated mapping algorithms for core allocation. All of them guarantee bounded expected tardiness and provide the same capacity augmentation bound; In practice, however, we expect them to provide different performances, both in terms of the task sets they can schedule and the actual tardiness they guarantee. Therefore, we performed numerical evaluations using randomly generated task sets to understand the practical differences between the three algorithms.

2014-48

Federated Scheduling for Stochastic Parallel Real-time Tasks

Authors: Jing Li, Kunal Agrawal, Christopher Gill, and Chenyang Lu

Corresponding Author: li.jing@wustl.edu

Abstract: Federated scheduling is a strategy to schedule parallel real-time tasks: It allocates a dedicated cluster of cores to high-utilization task (utilization >1); It uses a multiprocessor scheduling algorithm to schedule and execute all low-utilization tasks sequentially, on a shared cluster of the remaining cores. Prior work has shown that federated scheduling has the best known capacity augmentation bound of 2 for parallel tasks with implicit deadlines. In this paper, we explore the soft real-time performance of federated scheduling and address the average-case workloads instead of the worst-case values. In particular, we consider stochastic tasks -- tasks for which execution time and critical-path length are random variables. In this case, we use bounded expected tardiness as the schedulability criterion. We define a stochastic capacity augmentation bound and prove that federated scheduling algorithms guarantee the same bound of 2 for stochastic tasks. We present three federated mapping algorithms for core allocation. All of them guarantee bounded expected tardiness and provide the same capacity augmentation bound; In practice, however, we expect them to provide different performances, both in terms of the task sets they can schedule and the actual tardiness they guarantee. Therefore, we performed numerical evaluations using randomly generated task sets to understand the practical differences between the three algorithms.

Type of Report: Other

lel stochastic tasks that provide a soft real-time guarantee of bounded expected tardiness on uniform multicores: These algorithms provide a stochastic capacity bound of 2 for general DAG tasks. To our knowledge, this is the first result for stochastic parallel tasks. We also describe the procedure for calculating the corresponding (upper bound on) expected tardiness for all these algorithms.

2) Three different mapping algorithms for stochastic tasks: All these algorithms satisfy the same stochastic capacity augmentation bound and provide bounded tardiness. The three algorithms differ in their calculation for core allocation. They have increasing computation complexity (from linear-time to pseudo polynomial time) and also have increasing schedulability performance or expected tardiness.
3) A federated scheduling algorithm that uses *randomized work-stealing* scheduler [3] (instead of greedy scheduling) to schedule high utilization tasks: Work-stealing is a nearly-greedy, distributed, and randomized scheduling algorithm that is known to be more efficient than deterministic greedy schedulers in practice [4]; therefore, it may provide better overall efficiency to soft real-time applications.
4) We conduct numerical evaluations using randomly generated task sets to understand the efficacy of the different stochastic mapping algorithms.

The outline of the paper is as follows: Section II discusses related work. Section III defines stochastic task model and stochastic capacity augmentation bound. Section IV presents the stochastic federated scheduling strategy, expected tardiness calculation and prove that expected tardiness is bounded; Section V presents the three different mapping algorithms for core allocation; Section VI proves that these algorithms provide a capacity augmentation bound of 2; Section VII presents the soft real-time strategy that uses work-stealing.

II. Related Work

Real-time multiprocessor scheduling for **deterministic tasks** (with *worst-case* task parameters) has been studied extensively [5, 6]. In particular, for implicit deadline hard-real time deterministic tasks, the best known utilization bound is $\approx 50\%$ using partitioned fixed priority scheduling [7] and partitioned EDF [8]; this trivially implies a capacity augmentation bound of 2. In comparison, GEDF has a capacity augmentation bound of $2 - \frac{1}{m} + \epsilon$ for small ϵ [9, 10].

For parallel tasks with hard real-time constraints and worst-case task parameters, early work considered idealized models for tasks such as moldable and malleable tasks [11–14]. Most commonly considered model, recently, has been the **parallel synchronous model**, which is a subcategory of **directed acyclic graph (DAG)**. Many strategies for this model use task decomposition where parallel tasks are decomposed into a set of sequential tasks [1, 15–18]. Without decomposition, researchers have studied both synchronous tasks [19] and general DAG tasks [20–24]. For hard real-time tasks with worst-case parameters, best known capacity augmentation bound for general DAGs is 2 [2] using federated scheduling (partition-like strategy) without decomposition; 4 [24] using GEDF without decomposition (and was recently improved to 2.6 in an as yet unpublished result); 3.73 for general synchronous tasks [1]; and 3.42 [15] for a more restricted class of synchronous tasks.

Most prior work on bounded tardiness (and other soft real-time guarantees) considers deterministic sequential tasks with worst-case parameters [25]. For these tasks, earliest-pseudo-deadline-first scheduler [26] and GEDF [27, 28] both provide bounded tardiness with no utilization loss; these results were generalized to many global schedulers [29]. Lateness guarantees have also been studied for GEDF-like scheduling [30]. For parallel tasks, Liu [20] for the first time provide a soft real-time response time analysis for GEDF.

For stochastic analysis, there is some prior work on sequential stochastic tasks. For a resource reservation scheduler, a lower bound on the probability of deadline misses was derived in [31]. For multiprocessor scheduling, [32] shows that GEDF guarantees bounded tardiness to sequential tasks if the total expected utilization is smaller than the number of cores. We use this result directly in our algorithms and analysis to guarantee bounded tardiness to low-utilization tasks. There has also been some work on stochastic analysis of a system via Markov process or approximation [33, 34]. We are not aware of any work that considers stochastic parallel tasks.

There has been significant work on purely parallel systems, which are generally built to execute single parallel programs on pre-allocated cores to maximize throughput. Examples include parallel languages and runtime systems, such as the Cilk family [4, 35], OpenMP [36], and Intel's Thread Building Blocks [37]. While multiple tasks on a single platform have been considered in the context of fairness in resource allocation [38], none of this work considers real-time constraints.

III. Stochastic Parallel Task Model

In this section, we formalize the **stochastic task model** in which execution time and critical-path length are described using probabilistic distributions, which is consistent with the task model for sequential tasks in existing work on stochastic real-time analysis [32]. We also define the capacity augmentation bound for stochastic tasks with soft real-time tardiness constraint. Throughout this paper, we use the calligraphic letters to represent random variables.

Stochastic tasks have a fixed relative deadline D_i ($= P_i$, the period, for implicit deadline tasks). However, each stochastic task is described using its *stochastic work* $\mathcal{C}_i$ — execution time on 1 core, and *stochastic critical-path length* $\mathcal{L}_i$ — execution time on an infinite number of cores, where $\mathcal{C}_i$ and $\mathcal{L}_i$ are random variables. We assume that the expectations $\mathrm{E}[\mathcal{C}_i]$ and $\mathrm{E}[\mathcal{L}_i]$ of these random variables are known. Given these parameters, we can calculate the *expected utilization* of a stochastic task τ_i as $\mathrm{E}[\mathcal{U}_i] = \mathrm{E}[\mathcal{C}_i]/D_i$, and the total expected utilization of the entire task set as $\sum_i \mathrm{E}[\mathcal{U}_i]$.

The exact distributions of $\delta_{\mathcal{C}_i}$ and $\delta_{\mathcal{L}_i}$ are not explicitly required in all three schedulability tests. Our linear-time algorithm can calculate mappings that provide bounded tardiness using just these parameters. Providing the distributions, another algorithm can generate potentially better mappings.

We now specify a few additional parameters that are needed only if we wish to calculate an upper bound on the tardiness itself or to optimize this tardiness using our third (ILP-based) mapping algorithm. First, for all tasks, we must know the standard deviations $\delta_{\mathcal{C}_i}$ and $\delta_{\mathcal{L}_i}$ of the execution time and the critical-path length. Second, for low-utilization tasks, we need the finite worst-case execution time $\widehat{c}_i$ for calculating tardiness. Finally, for high-utilization tasks, we need the covariance $\sigma(\mathcal{C}_i, \mathcal{L}_i)$ between work and critical-path length.

In addition, for analysis purposes, we define some job specific parameters: $c_{i,j}$ is the actual execution time of the job j of task i and $l_{i,j}$ is its actual critical-path length; these are drawn from distributions $\mathcal{C}_i$ and $\mathcal{L}_i$ respectively. We say that the *release time* of job j of task i is $r_{i,j}$ and its *response time* (or *completion time*) is $t_{i,j}$. *Tardiness* $T_{i,j}$ of the job j of is defined as $\max(0, t_{i,j} - D_i)$. Tardiness $\mathcal{T}_i$ of a task τ_i is also a random variable; $\mathrm{E}[\mathcal{T}_i]$ is its expected value.

We now define the capacity augmentation bound for stochastic tasks. In particular, we consider the schedulability condition of **bounded expected tardiness**; that is, a task set τ is deemed schedulable by a scheduling algorithm $\mathcal{S}$ if the expected tardiness of each task is guaranteed to be bounded under $\mathcal{S}$.

Definition 1. *A scheduling algorithm $\mathcal{S}$ provides a **stochastic capacity augmentation bound** of b if, given m cores, $\mathcal{S}$ can guarantee bounded expected tardiness to any task set τ as long as it satisfies the following **conditions:***

$$\text{Total available cores, } m \geq b \sum \mathrm{E}[\mathcal{U}]_i \tag{1}$$

$$\text{For each task, } D_i \geq b(\mathrm{E}[\mathcal{L}]_i + \epsilon_i) \tag{2}$$

where ϵ_i is 0 if the variances of $\mathcal{C}_i$ and $\mathcal{L}_i$ are 0 and is an arbitrarily small positive constant otherwise.

Note that when $\mathcal{C}_i$ and $\mathcal{L}_i$ are deterministic, the variance of $\mathcal{C}_i$ and $\mathcal{L}_i$ is 0, so $\epsilon_i = 0$ and the definition of stochastic capacity augmentation bound reduces to the deterministic definition for hard real-time constraints.

IV. Stochastic Federated Scheduling Guarantees Bounded Tardiness

In this section, we firstly describe the stochastic federated scheduling; Secondly, we prove that if the federated scheduling can produce a mapping, then it guarantees bounded expected tardiness; Finally, we calculate the expected tardiness.

A. Stochastic Federated Scheduling Strategy

Just like the corresponding federated scheduling strategy for hard real-time tasks, the stochastic federated scheduling strategy classifies tasks into two sets: τ_{high} contains all **high-utilization tasks** — tasks with expected utilization at least 1 ($\mathrm{E}[\mathcal{U}_i] \geq 1$), and τ_{low} contains all the remaining **low-utilization tasks**. The federated scheduling strategy works in two stages:

1) Given a task set τ, a **mapping algorithm** either **admits** τ and outputs a core assignment, or declares that it cannot guarantee schedulability of τ. Different mapping algorithms differ in the assignment of n_i dedicated cores to each high-utilization task τ_i, but $n_i > \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]}$ is always required. All low-utilization tasks share the remaining $n_{\text{low}} = m - \sum_{\tau_i \in \tau_{high}} n_i$ cores. All the mapping algorithms only admit a task set, if $n_{\text{low}} > \sum_{\tau_i \in \tau_{\text{low}}} \mathrm{E}[\mathcal{U}_i]$ always holds.
2) Once the mapping is done, the scheduling is straightforward. The high-utilization tasks are scheduled on their dedicated cores using a greedy (work-conserving) scheduler. The low-utilization tasks are scheduled and executed sequentially on the remaining cluster of cores using GEDF scheduler.

Note that we chose GEDF to schedule low-utilization tasks, because we use an existing result that shows that GEDF provides bounded tardiness to sequential stochastic tasks [32]; we can directly apply this result to low-utilization tasks since they are executed sequentially by our federated scheduler. Other multiprocessor scheduling algorithms can be used only if they provide guarantees of bounded tardiness for sequential tasks.

B. Mapping Algorithms Guarantee Bounded Tardiness

We first analyze high-utilization tasks. Since each of them has dedicated cores and does not suffer any interference from other tasks, we can analyze each task τ_i individually. We use the following result from queueing theory [39] which indicates that if the service time of jobs is less than the inter-arrival time, then the expected waiting time is bounded.

Lemma 1. [KING70] *For a D/G/1 queue, customers arrive with minimum inter-arrival time Y, and the service time $\mathcal{X}$ is a distribution with mean $\mathrm{E}[\mathcal{X}]$ and variance $\delta^2_{\mathcal{X}}$. If $\mathrm{E}[\mathcal{X}] < Y$, then the queue is stable and the expected waiting time $\mathcal{W}$ is bounded $\mathrm{E}[\mathcal{W}] \leq \frac{\delta^2_{\mathcal{X}}}{2(Y - \mathrm{E}[\mathcal{X}])}$.*

In our context, for each high-utilization task, jobs are the customers; the inter-arrival time is $Y = D_i$ $(= P_i)$; the response time $t_{i,j}$ is the service time for job j of task τ_i. For a high-utilization job $\tau_{i,j}$, its tardiness $T_{i,j}$ depends on its response time $t_{i,j}$, the tardiness $T_{i,j-1}$ of previous job $\tau_{i,j-1}$ and deadline D_i. In particular, we have $T_{i,j+1} \leq \max\{0, T_{i,j-1} + t_{i,j} - D_i\}$. Therefore, the waiting time $\mathcal{W}$ is a bound on the tardiness $\mathcal{T}$.

For a greedy scheduler on n_i cores, there are two straightforward lemmas (Lemma 1 and 2) derived in [24]. Using the two Lemmas, we can easily bound the finish time $t_{i,j}$.

Lemma 2. *If a job $J_{i,j}$ executes by itself under a greedy scheduler on n_i identical cores and it takes $t_{i,j}$ time to finish its execution, then $t_{i,j} \leq (c_{i,j} + (n_i - 1)l_{i,j})/n_i$.*

Hence, the service time for a job is bounded by $(c_{i,j} + (n_i - 1)l_{i,j})/n_i$. Using properties of mean and variance, we get

$$\mathrm{E}[\mathcal{X}] = (\mathrm{E}[\mathcal{C}_i] + (n_i - 1)\mathrm{E}[\mathcal{L}_i])/ni \tag{3}$$

$$\delta^2_{\mathcal{X}} = \delta^2_{\mathcal{L}_i}((n_i - 1)/n_i)^2 + \delta^2_{\mathcal{C}_i}/n_i^2 + 2\sigma(\mathcal{L}_i, \mathcal{C}_i)(n_i - 1)/n_i^2 \tag{4}$$

Note that Lemma 1 states that if $\mathrm{E}[\mathcal{X}] < Y$, then the queue is stable and the tardiness is bounded. Therefore, to prove the bounded expected tardiness of high-utilization task, we only need to prove $\mathrm{E}[\mathcal{X}] = (\mathrm{E}[\mathcal{C}_i] + (n_i - 1)\mathrm{E}[\mathcal{L}_i])/n_i < D_i = Y$.

Theorem 1. *A mapping algorithm of stochastic federated scheduling guarantees bounded tardiness to high-utilization task τ_i, if the assigned number of cores $n_i > \frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]}{D_i-\mathrm{E}[\mathcal{L}_i]}$.*

Proof: We first prove $(\mathrm{E}[\mathcal{C}_i] + (n_i-1)\mathrm{E}[\mathcal{L}_i])/n_i < D_i$.

$$\begin{aligned} & D_i n_i - (n_i-1)\mathrm{E}[\mathcal{L}_i] = n_i(D_i - \mathrm{E}[\mathcal{L}_i]) + \mathrm{E}[\mathcal{L}_i] \\ > \quad & \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]}(D_i - \mathrm{E}[\mathcal{L}_i]) + \mathrm{E}[\mathcal{L}_i] = \mathrm{E}[\mathcal{C}_i] \end{aligned}$$

Hence, $\mathrm{E}[\mathcal{X}] = (\mathrm{E}[\mathcal{C}_i] + (n_i-1)\mathrm{E}[\mathcal{L}_i])/n_i < D_i = Y$ and by Lemma 1 the tardiness of τ_i is bounded. □

In the stochastic federated scheduling strategy, $n_i > \frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]}{D_i-\mathrm{E}[\mathcal{L}_i]}$ is always required for any mapping algorithm. We will show later that for all three proposed mapping algorithms, it is indeed satisfied for high-utilization task.

Now we analyze the tardiness of low-utilization tasks, since they share n_{low} cores and are executed sequentially using GEDF scheduler. In [32], the following Lemma has been established.

Lemma 3. [Mills10] *If a set of sequential tasks τ_{low} is scheduled on n_{low} cores using GEDF and $n_{low} > \sum_{\tau_i \in \tau_{low}} \mathrm{E}[\mathcal{U}_i]$, then the expected tardiness of each task is bounded.*

Since all the different mapping algorithms only admit a task set if $\mathrm{E}[\mathcal{U}_{\text{low}}] = \sum_{\tau_i \in \tau_{\text{low}}} \mathrm{E}[\mathcal{U}_i] < n_{\text{low}}$ and then schedules these tasks using GEDF, we can conclude that the expected tardiness of low-utilization tasks is also bounded.

Any task set that the mapping algorithm admits can be scheduled while guaranteeing bounded expected tardiness; hence, the mapping algorithm serves as a **schedulability test**.

C. Calculating Expected Tardiness

Here, we explain how the tardiness is calculated. Even though all the mapping algorithms provide bounded expected tardiness, the actual (upper bound on) tardiness can be different, because the corresponding core assignments (n_i for each high-utilization task and n_{low} for all low-utilization tasks) are different.

Note that from Section V, we can see that for BASIC and FAIR mapping algorithm, the tardiness calculation is not necessary for producing core assignment. It is only needed in ILP mapping or to actually get the expected tardiness.

1) Tardiness of High-Utilization Tasks: For each high-utilization tasks with n_i assigned dedicated cores, by Corollary 1 and Inequality (4), the bounded expected tardiness is:

$$\begin{aligned} & \mathrm{E}[\mathcal{T}_i] \le \frac{\delta_{\mathcal{X}}^2}{2(Y - \mathrm{E}[\mathcal{X}])} \\ & \le \frac{\delta_{\mathcal{L}_i}^2 (n_i-1)^2/n_i^2 + \delta_{\mathcal{C}_i}^2/n_i^2 + 2\sigma(\mathcal{L}_i,\mathcal{C}_i)(n_i-1)/n_i^2}{2(D_i - (\mathrm{E}[\mathcal{L}_i](n_i-1) + \mathrm{E}[\mathcal{C}_i])/n_i)} \end{aligned} \quad (5)$$

2) Tardiness of Low-Utilization Tasks: Since low-utilization tasks are executed sequentially using GEDF, we can use the linear-programming procedure described in [32] directly.

We first restate a couple of lemmas from [32] in our terminology. The first lemma bounds the tardiness of a hypothetical processor-sharing (**PS**) scheduler which always guarantees an execution rate of $\hat{u}_i$ (henceforth called the **PS rate allocation**) to each task τ_i.

Lemma 4. [Mills10] *For a given PS rate allocation such that $\mathrm{E}[\mathcal{U}_i] \le \hat{u}_i \le 1$ and $\sum \mathrm{E}[\mathcal{U}_i] \le n_{low}$, PS scheduler has a bounded tardiness $\mathrm{E}[\mathcal{F}_i] \le \frac{\delta_{\mathcal{C}_i}^2/\hat{u}_i^2}{2(D_i - \mathrm{E}[\mathcal{C}_i]/\hat{u}_i)}$.*

Using this PS tardiness bound, they can then provide a bound on the tardiness provided by GEDF for low-utilization tasks.

Lemma 5. [Mills10] *For low-utilization tasks scheduled by GEDF scheduler on n_{low} cores, the expected tardiness of each task $\mathrm{E}[\mathcal{T}_i] \le \mathrm{E}[\mathcal{F}_i] + \frac{\eta + n_{low} M}{n_{low} - v} + \hat{c}_i$, where $\mathrm{E}[\mathcal{F}_i]$ is the expected tardiness of a hypothetical PS scheduler, $\hat{c}_i$ is the worst-case execution time of the task, η is the sum of the $n_{low} - 1$ largest $\hat{c}_i$, M is the maximum tardiness in PS, and v is the sum of $n_{low} - 1$ largest assigned $\hat{u}_i$ in PS.*

All the parameters except $\mathrm{E}[\mathcal{F}_i]$ are known or measurable (and bounded). In order to calculate $\mathrm{E}[\mathcal{F}_i]$, we must calculate the PS rate allocation $\hat{u}_i$ for each task τ_i.

As will show in Section V, for BASIC mapping, there exists a simple calculation of $\hat{u}_i$; while for FAIR and ILP mappings, the following linear program (LP) from [32] (can be derived using Lemma 4) is used to calculate the PS rate allocations.

$$\begin{aligned} \max \quad & \zeta \\ \text{s.t.} \quad & D_i \hat{u}_i - \frac{\delta_{\mathcal{C}_i}^2}{2}\zeta \ge \mathrm{E}[\mathcal{C}_i] \quad \forall i, \mathrm{E}[\mathcal{U}_i] < 1 \\ & \sum_{i, \mathrm{E}[\mathcal{U}_i]<1} \hat{u}_i \le \hat{n}_{\text{low}} \\ & u_i \le \hat{u}_i \le 1 \quad \forall i, \mathrm{E}[\mathcal{U}_i] < 1 \end{aligned}$$

where $\zeta^{-1} \ge \max_i \left(\frac{\delta_{\mathcal{L}_i}^2}{2(\hat{u}_i D_i - \mathrm{E}[\mathcal{L}_i])}\right) = \max_i \mathrm{E}[\mathcal{F}_i]$. Therefore, solving the linear program provides us with the PS rate allocations $\hat{u}_i$ as well as a bound on the expected tardiness $\mathrm{E}[\mathcal{F}_i]$ of PS scheduler. Given these values, we can calculate the tardiness of low-utilization tasks using Lemma 5.

V. Mapping Algorithms for Stochastic Federated Scheduling

We propose three federated mapping algorithms for stochastic federated scheduling. The three algorithms differ in their calculation of n_i for high-utilization tasks. They have increasing computation complexity and also have increasing schedulability performance or expected tardiness: The first algorithm, **BASIC**, assigns cores based on utilization; The second algorithm, **FAIR**, assumes that the distribution of execution time and critical-path length is known, and it assigns cores based on the values with same cumulative possibility from task parameter distributions among all tasks; The last ILP-Based algorithm, (**ILP**), tries to minimize the maximum expected tardiness.

A. BASIC Stochastic Federated Mapping Algorithm

For a high-utilization tasks τ_i, this mapping algorithm calculates n_i, the number of cores assigned to τ_i as follows:

$$n_i = \begin{cases} \left\lceil \frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]-\alpha_i}{D_i-\mathrm{E}[\mathcal{L}_i]-\alpha_i} \right\rceil & (\mathrm{E}[\mathcal{U}_i] > 1) \\ 2 & (\mathrm{E}[\mathcal{U}_i] = 1) \end{cases} \quad (6)$$

where $\alpha_i = D_i/b - \mathrm{E}[\mathcal{L}_i] > 0$ and $b = 2$.

The remaining $n_{\text{low}} = m - \sum_{\text{high}} n_i$ cores are assigned to the low-utilization tasks. The mapping algorithm admits a task set as long as $\mathrm{E}[\mathcal{U}_{\text{low}}] = \sum_{\text{low}} \mathrm{E}[\mathcal{U}_i] \leq n_{\text{low}}/b$ for $b = 2$.

Note that the major difference between this n_i and the one in [2] is the extra term α_i. α_i is used to accommodate the variation of execution time and critical-path length. We set this value of α_i to assign roughly same number of cores relative to utilization. Hence, variances are not required to assign cores.

Bounded Tardiness (Schedulability Test): The tardiness can be bounded for any positive α_i since: For $\mathrm{E}[\mathcal{U}_i] = 1$, $\frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]}{D_i-\mathrm{E}[\mathcal{L}_i]} = 1$, so $n_i = 2 > \frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]}{D_i-\mathrm{E}[\mathcal{L}_i]}$. For $\mathrm{E}[\mathcal{U}_i] > 1$,

$$n_i \geq \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} > \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} > 1, \text{ since}$$

$D_i - \mathrm{E}[\mathcal{L}_i] > \alpha_i > 0$. Also, $\mathrm{E}[\mathcal{U}_{\text{low}}] \leq n_{\text{low}}/2 < n_{\text{low}}$. By Theorem 1 and Lemma 3, BASIC can guarantee bounded tardiness for both high and low-utilization tasks. Therefore, the BASIC serves as a schedulability test that runs in linear time.

Tardiness calculation: Now we describe a faster and simpler method to calculate the upper bound on the expected tardiness of low-utilization tasks when using BASIC mapping. This method relies on the requirement of BASIC that $n_{\text{low}} \geq b\sum_{\text{low}} \mathrm{E}[\mathcal{U}_i]$ for b =2. We can simply set PS rate allocation for a task τ_i as $\hat{u}_i = \min(b\mathrm{E}[\mathcal{U}_i], 1)$. This allocation satisfies the requirement in Lemma 4; therefore, the PS tardiness is

$$\mathrm{E}[\mathcal{F}_i] \leq \frac{\delta^2_{\mathcal{C}_i}}{2(\hat{u}_i^2 D_i - \hat{u}_i \mathrm{E}[\mathcal{C}_i])},$$

and by Lemma 5 the expected tardiness of low-utilization task under GEDF can be calculated directly as

$$\mathrm{E}[\mathcal{T}_i] \leq \frac{\delta^2_{\mathcal{C}_i}}{2(\hat{u}_i^2 D_i - \hat{u}_i \mathrm{E}[\mathcal{C}_i])} + \frac{\eta + n_{\text{low}} M}{n_{\text{low}} - v} + \hat{e}_i, \tag{7}$$

Therefore, unlike the FAIR and ILP algorithms, tardiness calculation here does not require solving a linear program; it can be done in linear time.

B. FAIR Federated Mapping Algorithm

We now present FAIR mapping that admits more task sets than the BASIC, while still providing same theoretical guarantees. The schedulability test of FAIR still runs in linear time; however, the calculations of the core assignment and the expected tardiness are more complex, requiring near linear time and linear programming respectively.

We donate $\mathcal{C}_i(p)$ as the value c_i of random variables $\mathcal{C}_i$ when its cumulative distribution function (CDF) $\mathbb{F}_{\mathcal{C}_i}(c_i) = p$ (meaning that the possibility of $\mathcal{C}_i \leq c_i$ is equal to p). We denote $\mathcal{L}_i(p)$ and $\mathcal{U}_i(p)$ similarly.

Note that when $p = 0.5$, $\mathcal{C}_i(p) = \mathrm{E}[\mathcal{C}_i]$ and $\mathcal{L}_i(p) = \mathrm{E}[\mathcal{L}_i]$. Additionally, $\mathcal{C}_i(p)$ and $\mathcal{L}_i(p)$ will increase when p increases.

In FAIR mapping, the number of cores assigned to high-utilization task τ_i (represented by $\hat{n}_i$) is calculated as follows.

$$\hat{n}_i(0.5 \leq p < 1) = \left\lfloor \frac{\mathcal{C}_i(p) - \mathcal{L}_i(p)}{D_i - \mathcal{L}_i(p)} + 1 \right\rfloor \tag{8}$$

$$= \begin{cases} \left\lceil \frac{\mathcal{C}_i(p)-\mathcal{L}_i(p)}{D_i-\mathcal{L}_i(p)} \right\rceil & \left(\frac{\mathcal{C}_i(p)-\mathcal{L}_i(p)}{D_i-\mathcal{L}_i(p)} \text{ is not integer}\right) \\ \frac{\mathcal{C}_i(p)-\mathcal{L}_i(p)}{D_i-\mathcal{L}_i(p)} + 1 & \left(\frac{\mathcal{C}_i(p)-\mathcal{L}_i(p)}{D_i-\mathcal{L}_i(p)} \text{ is integer}\right) \end{cases}$$

FAIR mapping will admit a task set if $n_{\text{low}} = m - \sum_{\text{high}} \hat{n}_i(p) > \sum_{\text{low}} \mathrm{E}[\mathcal{U}_i(p)]$ for $p = 0.5$.

Bounded Tardiness (Schedulability Test): It is obvious that $\hat{n}_i(p = 0.5) = \left\lfloor \frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]}{D_i-\mathrm{E}[\mathcal{L}_i]} + 1 \right\rfloor > \frac{\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i]}{D_i-\mathrm{E}[\mathcal{L}_i]}$. Also $n_{\text{low}} > \sum_{\text{low}} \mathrm{E}[\mathcal{U}_i(p = 0.5)] = \sum_{\text{low}} \mathrm{E}[\mathcal{U}_i]$. Then by Theorem 1 and Lemma 3, FAIR guarantees bounded tardiness for all tasks. The FAIR also serves as a linear time schedulability test.

Dominance in Schedulability: In Section VI, we will show that $\hat{n}_i(p = 0.5) \leq n_i$ (of BASIC mapping) for any task τ_i and hence $\hat{n}_{low} \geq n_{low}$. Also, the FAIR algorithm allows $\mathrm{E}[\mathcal{U}_{\text{low}}]$ to be as high as $\hat{n}_{low}$ (instead of $n_{low}/2$ allowed by BASIC). Therefore, FAIR admits strictly more tasks than BASIC.

Core Allocation: $\hat{n}_i(p = 0.5)$, name as *minimum core assignment*, is the minimum number of cores required to guarantee bounded tardiness for high-utilization tasks. However, directly using it will result in large tardiness for high-utilization tasks, because more cores are assigned to low-utilization tasks. To be fair to all tasks, FAIR mapping further improve the *minimum core allocation* by increasing p until the largest $\hat{p}$ when $n_{\text{low}} = m - \sum_{\text{high}} \hat{n}_i(\hat{p}) > \sum_{\text{low}} (\mathcal{C}_i(\hat{p})/D_i)$. By doing this, FAIR in fact increase the core assignment and PS rate allocation for each task by the same amount according to the CDF of execution time and critical-path length. This ensures fairness among all tasks, because $\hat{p}$ is independent of τ_i. The complexity of this core assignment depends on the number of p tested until reaching $\hat{p}$. In practice, a binary search will only need 6 times at most to find $\hat{p}$ with an accuracy of 0.01.

C. ILP-Based Federated Mapping Algorithm

We now present a third ILP-Based mapping algorithm for stochastic federated scheduling. This algorithm admits exactly the same task sets as FAIR (though it may find a different mapping for these task sets); therefore, it also provides the same theoretical guarantees. However, BASIC and FAIR make no attempt explicitly to balance maximum tardiness among high and low-utilization tasks.

The ILP algorithm converts the mapping problem for high-utilization tasks into a *integer linear program* (ILP) that tries to minimize the maximum tardiness; When combined with the linear program for low-utilization tasks stated in Section IV-C2, the resulting mixed linear program indirectly tries to balance the tardiness among all tasks.

We convert Inequality (5) into a form similar to the expected tardiness of the PS schedule; that is, we define ζ_i where $\zeta^{-1} = \max_i \mathrm{E}[\mathcal{T}_i]$ and ζ is defined in terms of n_i's. First, for task τ_i, let $\delta_i^2 = max\big(\delta^2_{\mathcal{L}_i}(m-1)^2/m, \delta^2_{\mathcal{C}_i}/2, \sigma(\mathcal{L}_i, \mathcal{C}_i)(m-1)/m\big)$. Note that, δ_i^2 is bounded and can be calculated using only the expectation and variance of the task's execution time and critical-path length without knowing n_i. Now we use the fact that $2 \leq n_i \leq m$ for high-utilization task τ_i and see that

$$\begin{aligned}
\delta_i^2 &\geq \delta^2_{\mathcal{L}_i}(m-1)^2/m = \delta^2_{\mathcal{L}_i}(m-1)(1-1/m) \\
&\geq \delta^2_{\mathcal{L}_i}(n_i-1)(1-1/n_i) = \delta^2_{\mathcal{L}_i}(n_i-1)^2/n_i \\
\delta_i^2 &\geq \delta^2_{\mathcal{C}_i}/2 \geq \delta^2_{\mathcal{C}_i}/n_i \\
\delta_i^2 &\geq \sigma(\mathcal{L}_i, \mathcal{C}_i)(m-1)/m = \sigma(\mathcal{L}_i, \mathcal{C}_i)(1-1/m) \\
&\geq \sigma(\mathcal{L}_i, \mathcal{C}_i)(1-1/n_i) = \sigma(\mathcal{L}_i, \mathcal{C}_i)(n_i-1)/n_i.
\end{aligned}$$

Now we calculate the upper bound on the variance of $\delta_{\mathcal{X}}^2$ (from Inequality (4)) using δ_i^2

$$\begin{aligned}\delta_{\mathcal{X}}^2 &= \delta_{\mathcal{L}_i}^2(n_i-1)^2/n_i^2 + \delta_{\mathcal{C}_i}^2/n_i^2 + 2\sigma(\mathcal{L}_i,\mathcal{C}_i)(n_i-1)/n_i^2 \\ &= \frac{\delta_{\mathcal{L}_i}^2(n_i-1)^2/n_i + \delta_{\mathcal{C}_i}^2/n_i + 2\sigma(\mathcal{L}_i,\mathcal{C}_i)(n_i-1)/n_i}{n_i} \\ &\leq 4\delta_i^2/n_i\end{aligned}$$

By Corollary 1, the expected tardiness is bounded by

$$\begin{aligned}\mathrm{E}[\mathcal{T}_i] &\leq \frac{\delta_{\mathcal{X}}^2}{2(Y-\mathrm{E}[\mathcal{X}])} \\ &\leq \frac{4\delta_i^2/n_i}{2(D_i-(\mathrm{E}[\mathcal{L}_i](n_i-1)+\mathrm{E}[\mathcal{C}_i])/ni)} \\ &\leq \frac{2\delta_i^2}{n_iD_i-(\mathrm{E}[\mathcal{L}_i](n_i-1)+\mathrm{E}[\mathcal{C}_i])} \\ &= \frac{2\delta_i^2}{n_i(D_i-\mathrm{E}[\mathcal{L}_i])-(\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i])} \qquad (9)\end{aligned}$$

Now we can set $\zeta^{-1} \geq \max_i(\frac{2\delta_i^2}{n_i(D_i-\mathrm{E}[\mathcal{L}_i])-(\mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i])}) \geq \max_i \mathrm{E}[\mathcal{T}_i]$ for high-utilization tasks and get inequality (11).

Combining this definition of ζ with the linear program in Section IV-C2, we get the following mixed linear program:

$$\begin{aligned}\max \quad & \zeta \\ \text{s.t.} \quad & D_i\hat{u}_i - \frac{\delta_{\mathcal{C}_i}^2}{2}\zeta \geq \mathrm{E}[\mathcal{C}_i] && \forall i, \mathrm{E}[\mathcal{U}_i] < 1 \quad (10) \\ & (D_i-\mathrm{E}[\mathcal{L}_i])n_i - 2\delta_i^2\zeta \geq \mathrm{E}[\mathcal{C}_i]-\mathrm{E}[\mathcal{L}_i] && \forall i, \mathrm{E}[\mathcal{U}_i] \geq 1 \quad (11) \\ & \sum_{i,\mathrm{E}[\mathcal{U}_i]<1}\hat{u}_i + \sum_{i,\mathrm{E}[\mathcal{U}_i]\geq 1} n_i \leq m && \quad (12) \\ & u_i \leq \hat{u}_i \leq 1 && \forall i, \mathrm{E}[\mathcal{U}_i] < 1 \quad (13) \\ & \hat{n}_i(p=0.5) \leq n_i && \forall i, \mathrm{E}[\mathcal{U}_i] \geq 1 \quad (14) \\ & n_i \text{ is integer} && \forall i, \mathrm{E}[\mathcal{U}_i] \geq 1 \quad (15)\end{aligned}$$

We solve this ILP to calculate: integral n_i— the number of cores assigned to high utilization task τ_i; fractional $\hat{u}_i$ — a valid PS rate allocation for low-utilization task τ_i; and ζ. Using the resulting n_i for high utilization tasks, we can calculate $n_{\text{low}} = m - \sum_{\text{high}} n_i$, the number of cores assigned to low-utilization tasks.

Explanation of Constraints: Constraints (14) and (15) guarantee that each high-utilization task τ_i gets at least $\hat{n}_i(p=0.5)$ dedicated cores; therefore Theorem 1 guarantees its bounded tardiness. Constraint (13) guarantees that the PS rate allocation is larger than the utilization of low-utilization tasks; therefore by Lemma 4 guarantees bounded tardiness to these tasks. Constraint (12) guarantees that $n_{\text{low}} + n_{\text{high}} \leq m$. Finally, Constraint (10) is inherited from the LP in Section IV-C2.

Optimal Greedy Solution to the ILP: General ILP problem can be hard to solve. However, there is a unique property of the above ILP — ζ will decrease if at least one n_i or $\sum_{\text{low}} \hat{u}_i$ increase and the rest ones remain the same. Relying on this, we can easily see that a greedy algorithm — starting with the core assignment (n_i and $\hat{u}_i(p=0.5)$) from the minimum core allocation of FAIR mapping, iteratively increase the one n_i or $\sum_{\text{low}} \hat{u}_i$ (a high utilization task or the sum of low utilization tasks) with largest tardiness by 1 and stop when Constraint (12) will not hold — will successfully find the optimal solution to this ILP problem (providing the fact that the LP in Section IV-C2 can directly calculate optimal solution). By applying the greedy solution, we can reduce the mixed-ILP problem to a iterative LP problem. Obviously. the maximum number of iterations needed by the greedy algorithm is m.

Relationship to FAIR: The ILP mapping algorithm admits exactly the same task sets that FAIR admits; If the FAIR admits a task set ($\hat{n}_i(p=0.5)$; $n_{\text{low}} = m - \sum_{\text{high}} \hat{n}_i(p=0.5)$), then that mapping is a trivially feasible solution to the ILP since it satisfies all constraints for $\zeta = 0$. On the other hand, if the FAIR algorithm cannot find a solution, then there is no feasible solution to the ILP. Therefore, since FAIR provides a capacity augmentation bound of 2, so does this algorithm.

Faster Schedulability Test: As a consequence of the relationship with FAIR, we do not have to solve the ILP to check if the task set is schedulable using this ILP-based mapping; we can simply run the schedulability test of FAIR to check for schedulability and only solve the ILP to find the mapping if the task set is, in fact, schedulable.

Tardiness Calculation: On solving the mixed linear program, we get n_i for each high utilization task and $\hat{u}_i$ for each low utilization task. Therefore, we can use Inequalities (5) and (7) to calculate the tardiness of these tasks, respectively.

Note that, the mixed linear program criterion is a little imprecise; maximizing ζ does not directly optimize the overall tardiness bound. Instead, it only tries to balance parts of the tardiness. After applying the Inequalities (7) and (5) for calculating tardiness, the resulting tardiness of high-utilization tasks is actually less than the optimized bound ζ^{-1}, while the tardiness of low-utilization tasks is actually higher than ζ^{-1}.

To further balance the overall tardiness, instead of using the strict upper bound of $\delta_{\mathcal{X}}^2$ (from Inequality (9)) in the calculation of ζ, we can approximate it. The reason we cannot directly use Inequality (4) to calculate $\delta_{\mathcal{X}}^2$ is because we do not know n_i before we solve the integer linear program. However, we can approximate $\delta_{\mathcal{X}}^2$ by using $\hat{n}_i\hat{n}_i(p=0.5)$ instead of n_i. Then, we have $\delta_{\mathcal{X}}^2 = \frac{\delta_{\mathcal{L}_i}^2(\hat{n}_i-1)^2/\hat{n}_i + \delta_{\mathcal{C}_i}^2/\hat{n}_i + 2\sigma(\mathcal{L}_i,\mathcal{C}_i)(\hat{n}_i-1)/\hat{n}_i}{n_i} = \frac{\delta_i^2}{n_i}$. This may provide a better tardiness bound for all tasks.

However, when the worst-case execution time of a low-utilization task is large, the achieved mapping may still result in a larger maximum tardiness (from that task) than the optimal.

VI. Stochastic Capacity Augmentation Bound of 2 for Stochastic Federated Scheduling

A. Stochastic Capacity Augmentation Bound for BASIC

Theorem 2. *The BASIC federated scheduling algorithm has a stochastic capacity augmentation bound of b =2.*

In order to prove Theorem 2, we first prove that the BASIC mapping strategy always admits all *eligible* task sets — task sets that satisfy Conditions (1) and (2) in Definition 1 for b =2.

BASIC admits a task set if, $\mathrm{E}[\mathcal{U}_{\text{low}}] \leq n_{\text{low}}/b$ for $b = 2$. Therefore, we must prove that for all task sets that satisfy

Conditions (1) and (2), n_{low} is large enough for BASIC to admit the task set.

First, we prove that the number of cores assigned to high-utilization tasks n_{high} is bounded by $b\sum_{\text{high}} \mathrm{E}[\mathcal{U}_i]$.

Lemma 6. *For a high-utilization task τ_i ($1 \le \mathrm{E}[\mathcal{U}_i]$), if $D_i > b\mathrm{E}[\mathcal{L}]_i$ (Condition (2)), then the number of assigned cores $n_i \le b\mathrm{E}[\mathcal{U}_i]$ with $b = 2$.*

Proof: For $\mathrm{E}[\mathcal{U}_i] > 1$, since $b(\mathrm{E}[\mathcal{L}_i] + \alpha_i) = D_i$, so $\mathrm{E}[\mathcal{C}_i] = b(\mathrm{E}[\mathcal{L}_i]+\alpha_i)\mathrm{E}[\mathcal{U}_i]$ and $D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i = (b-1)(\mathrm{E}[\mathcal{L}_i]+\alpha_i)$.

$$n_i = \left\lceil \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} \right\rceil < \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} + 1$$
$$= \frac{2(\mathrm{E}[\mathcal{L}_i] + \alpha_i)\mathrm{E}[\mathcal{U}_i] - (\mathrm{E}[\mathcal{L}_i] + \alpha_i)}{\mathrm{E}[\mathcal{L}_i] + \alpha_i} + 1 = 2\mathrm{E}[\mathcal{U}_i]$$

For $\mathrm{E}[\mathcal{U}_i] = 1$, $n_i = 2 = 2\mathrm{E}[\mathcal{U}_i]$. Therefore, $n_{\text{high}} = \sum_{\text{high}} n_i \le b\sum_{\text{high}} \mathrm{E}[\mathcal{U}_i]$ for $b = 2$. □

Since the task set τ satisfies Condition (1), the total utilization $\sum \mathrm{E}[\mathcal{U}_i] \le m/b$ for b=2. So we have $n_{\text{low}} = m - n_{\text{high}} \ge b\sum_i \mathrm{E}[\mathcal{U}_i] - b\sum_{\text{high}} \mathrm{E}[\mathcal{U}_i] = b\sum_{\text{low}} \mathrm{E}[\mathcal{U}_i]$. Hence, BASIC's admission criterion is satisfied and it admits any task set satisfying Conditions (1) and (2). Since BASIC always provides bounded tardiness to task sets it admits (Section IV-B), by Definition 1 this establishes Theorem 2.

B. Stochastic Capacity Augmentation Bound for FAIR

Theorem 3. *The FAIR federated scheduling algorithm has a stochastic capacity augmentation bound of b =2.*

To prove Theorem 3, we simply prove if the BASIC admits a task set, then FAIR does as well; since BASIC admits any task set that satisfies Conditions (1) and (2) of Definition 1 for b =2, FAIR also admits them. Since FAIR always provides bounded tardiness to task sets it admits, this establishes Theorem 3.

First, we show that the minimum core assignment $\hat{n}_i(p = 0.5)$ to each high-utilization task by the FAIR algorithm is at most the number of cores n_i that the BASIC algorithm assigns.

Lemma 7. *If $\hat{n}_i = \hat{n}_i(p = 0.5) = \left\lfloor \frac{\mathcal{C}_i(p) - \mathcal{L}_i(p)}{D_i - \mathcal{L}_i(p)} + 1 \right\rfloor = \left\lfloor \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} + 1 \right\rfloor$; and $n_i = \left\lceil \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} \right\rceil$ for $\mathrm{E}[\mathcal{U}_i] > 1$ and $n_1 = 2$ for $\mathrm{E}[\mathcal{U}_i] = 1$; then $\hat{n}_i \le n_i$.*

Proof: To make the proof straightforward, now we use the two cases definition of $\hat{n}_i$ in Section V.

For $\mathrm{E}[\mathcal{U}_i] > 1$, obviously $\frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} > \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} > 1$, since $D_i - \mathrm{E}[\mathcal{L}_i] > \alpha_i > 0$. So we denote $\frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} = \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} + \epsilon$, so $\epsilon > 0$. When $\frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]}$ is not integer,

$$\hat{n}_i = \left\lceil \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} \right\rceil \le \left\lceil \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} \right\rceil = n_i$$

When $\frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]}$ is integer, since $\epsilon > 0$,

$$n_i = \left\lceil \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i] - \alpha_i}{D_i - \mathrm{E}[\mathcal{L}_i] - \alpha_i} \right\rceil = \left\lceil \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} + \epsilon \right\rceil$$
$$\ge \frac{\mathrm{E}[\mathcal{C}_i] - \mathrm{E}[\mathcal{L}_i]}{D_i - \mathrm{E}[\mathcal{L}_i]} + 1 = \hat{n}_i$$

For $\mathrm{E}[\mathcal{U}_i] = 1$, $\hat{n}_i = 2 = n_i$. Therefore, for all cases, $\hat{n}_{\text{high}} = \sum_{\text{high}} \hat{n}_i \le \sum_{\text{high}} n_i = n_{\text{high}}$. □

FAIR has more cores available for low utilization tasks than BASIC does, since $\hat{n}_{\text{low}} = m - \hat{n}_{\text{high}} \ge m - n_{\text{high}} = n_{\text{low}}$. It also allows the total utilization of low-utilization tasks to be as high as n_{low}, while basic only allows it to be n_{low}/b. Therefore, FAIR admits any task set that BASIC admits.

Note that FAIR will only increase $\hat{n}_i$ to $\hat{n}_i(\hat{p})$ if it can admits the task set. Therefore, as far as schedulability and capacity augmentation bound is concerned, this will not affect the proof above. In this most loaded cases, $\hat{n}_i(\hat{p}) = \hat{n}_i(p = 0.5)$.

VII. Work-Stealing for High-Utilization Tasks

We now switch gears and analyze federated scheduling of stochastic tasks when high-utilization tasks are scheduled using *randomized work-stealing* scheduler [3] instead of a purely greedy scheduler. A (deterministic) purely greedy scheduler often has high overheads, since it must maintain some sort of centralized queue of available work and all cores potentially suffer contention when they access this queue to get work. In a real implementation, we would model these overheads by inflating the tasks' execution times, potentially decreasing the real efficiency of the platform.

As comparison, **work-stealing** is an approximation of greedy scheduling, which makes scheduling decision randomly in a distributed manner. Each thread maintains a local queue of ready work and takes work from this queue as needed. If a thread's local queue is empty, it randomly picks another thread (running on another core) and **steals** some work from its queue. Work-stealing is not a strictly greedy strategy. However, it provide strong probabilistic guarantees of linear speedup ("near-greediness") [40]. In practice, it has less overheads and provides good performance [4]. It is the default strategy used in many parallel computing runtime systems such as Cilk, Cilk Plus, TBB, X10, and TPL [4, 35, 37, 41, 42].

Since work-stealing is a randomized scheduler, the response time of tasks is always a random variable despite of using whether worst-case values or stochastic values; therefore, we can easily extend the machinery developed so far to analyze the expected tardiness bound for both types of tasks.

A. Work-Stealing for Tasks using Worst-Case Values

We denote the worst-case execution time as C_i and worst-case critical-path length as L_i for such tasks. Each high-utilization task $\tau_i \in \tau_{\text{high}}$ is assigned n_i cores:

$$n_i = \left\lfloor \frac{C_i}{D_i - 3.65L_i - 1} + 1 \right\rfloor \tag{16}$$

The remaining cores $n_{\text{low}} = m - \sum_{\text{high}} n_i$ are assigned to low-utilization tasks which are scheduled using multiprocessor GEDF scheduler. The work-stealing mapping strategy admits a task set as long as $n_{\text{low}} \ge \sum_{\text{low}} U_i$.

Proof of Bounded Tardiness: We first state known results on work-stealing response time γ_i for task τ_i with total execution time C_i and critical path-length L_i [40].

Lemma 8. [Tchi.13] *A work stealing scheduler guarantees a completion time of γ_i, such that*

$$\mathrm{E}[\gamma_i] \leq \frac{C_i}{n_i} + 3.65L_i + 1 \tag{17}$$

$$\mathbb{P}\left\{\gamma_i \geq \frac{C_i}{n_i} + 3.65(L_i + \log_2 \frac{1}{\epsilon}) + 1\right\} \leq \epsilon \tag{18}$$

This probability distribution has mean $\mu = \mathrm{E}[\gamma_i] = \frac{C_i}{n_i} + 3.65L_i + 1$. In order to calculate its standard deviation, we define the CDF of γ_i as $\mathbb{F}(x) = \mathbb{P}\{\gamma_i \geq x\}$. In addition, we pessimistically assume that $\mathbb{F}(x) = \epsilon$, i.e. the probability of a longer completion time than x is ϵ, which is larger than the real probability. By definition, we can calculate

$$x = \frac{C_i}{n_i} + 3.65(L_i + \log_2 \frac{1}{\epsilon}) + 1 = \mu + 3.65 \log_2 \frac{1}{\epsilon}$$

Therefore, for any ϵ, we get

$$\mathbb{F}(x) = \epsilon = 1 - e^{-\frac{\ln 2}{3.65}(x-\mu)};$$

This is a shifted exponential distribution with standard deviation of $\frac{\ln 2}{3.65}$. Therefore, the standard deviation of the completion time is no more than $\frac{\ln 2}{3.65} \approx 0.19$.

The tardiness of job j of task i is $T_{i,j} = \max\{0, T_{i,j-1} + \gamma_{i,j} - D_i\}$ where $\gamma_{i,j}$ is drawn from the distribution γ_i above. Therefore, Lemma 1 guarantees bounded tardiness if

$$\mathrm{E}[\mathcal{X}] = \mathrm{E}[\gamma_i] = \frac{C_i}{n_i} + 3.65L_i + 1 < D_i = Y$$

which is true for $n_i \geq \left\lfloor \frac{C_i}{D_i - 3.65L_i - 1} + 1 \right\rfloor$. Therefore, we have proven tardiness of high-utilization tasks since we assign $\left\lfloor \frac{C_i}{D_i - 3.65L_i - 1} + 1 \right\rfloor$ cores to them. The tardiness of low-utilization tasks is bounded due to Lemma 3.

Upper Bound on Tardiness: The tardiness of high-utilization tasks is computed using the D/G/1 queue described in Lemma 1. Given n_i cores, we can calculate the tardiness of a high-utilization task τ_i using Lemma 1.

$$\mathrm{E}[\mathcal{T}_i] \leq \frac{\delta_{\gamma_\}}^2}{2(Y - \mathrm{E}[\gamma_\])} \leq \frac{\left(\frac{\ln 2}{3.65}\right)^2}{2(D_i - (\frac{C_i}{n_i} + 3.65L_i + 1))}$$

B. Work-Stealing for Stochastic Tasks

We can also provide bounded tardiness to stochastic tasks in essentially the same manner. In particular, the mapping algorithm simply uses the expected values for work and critical path length, but otherwise is the same as Equation (16). We can calculate the mean and the standard deviation of $\mathcal{X}$ using essentially the same procedure as follows:

$$\begin{aligned}\mathrm{E}[\mathcal{X}] = \mathrm{E}[\mathrm{E}[\gamma_i]] &= \mathrm{E}[\mathcal{C}_i]/n_i + 3.65\mathrm{E}[\mathcal{L}_i] + 1 \\ \delta_{\mathcal{X}}^2 = \delta_{\gamma_i}^2 &= (\ln 2/3.65)^2 + \delta_{\mathcal{L}_i}^2 (n_i - 1)^2/n_i^2 \\ &\quad + \delta_{\mathcal{C}_i}^2/n_i^2 + 2\sigma(\mathcal{L}_i, \mathcal{C}_i)(n_i - 1)/n_i^2\end{aligned}$$

By Lemma 1, the tardiness of high-utilization tasks is

$$\begin{aligned}\mathrm{E}[\mathcal{T}_i] \quad \leq \quad & \frac{(0.19)^2 + \delta_{\mathcal{L}_i}^2 (n_i - 1)^2/n_i^2}{2(D_i - (\mathrm{E}[\mathcal{C}_i]/n_i + 3.65\mathrm{E}[\mathcal{L}_i] + 1))} \\ + \quad & \frac{\delta_{\mathcal{C}_i}^2/n_i^2 + 2\sigma(\mathcal{L}_i, \mathcal{C}_i)(n_i - 1)/n_i^2}{2(D_i - (\mathrm{E}[\mathcal{C}_i]/n_i + 3.65\mathrm{E}[\mathcal{L}_i] + 1))}\end{aligned}$$

Compared with greedy scheduler, the additional $2.65\mathrm{E}[\mathcal{L}_i]$ on the denominator is the major contribution to additional overhead. But if a task is highly paralleled, the performance of work-stealing is nearly the same as a greedy scheduler.

VIII. Numerical Evaluation

To compare the different performances of three schedulability tests for *stochastic task sets*, here, we present our numerical evaluation on randomely generated task sets with probability distribution on execution time and critical-path length.

A. Task Sets Generation and Experimental Setup

We evaluate the schedulability results on varying number of cores m: $4, 8, 16, 32$, and 64. For various total task set utilizations U starting from 10% to 80%, we generate task sets, add tasks and load the system to be exactly mU — fully loading a unit speed machine. Results of 4 and 64 cores are similar to the rest, so we omitted them due to space limit.

For each task, we assume a normal distribution of execution time and critical-path length. We uniformly generate the expected execution time $\mathrm{E}[\mathcal{C}_i]$ between 1 and 100. Then for tasks with small variance, we uniformly generate variance to be from 5% to 10% of $\mathrm{E}[\mathcal{C}_i]$; for task with large variance, we let it be from 5% to 500%. We generate the critical-path length following the same rules and ensure the average parallelism $\mathrm{E}[\mathcal{C}_i]/\mathrm{E}[\mathcal{L}_i]$ is 32. To ensure a reasonable amount of high-utilization tasks in a task set on m cores, we uniformly generate the task utilization u_i between 0.4 to $\sqrt{m}$. Since we assume a normal distribution for execution time and critical-path length, with the expected mean and standard deviation, we can calculate the worst-case execution time by calculating the value $\hat{c}_i$ of the distribution when the possibility of a longer execution time is less than 0.01. Deadline is calculated by $u_i \mathrm{E}[\mathcal{C}_i]$

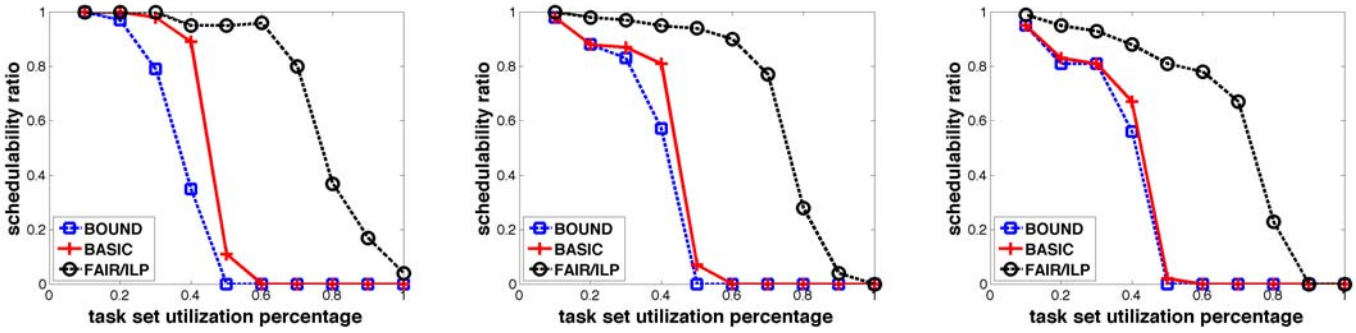
Using the task set setups above, we run each setting for 100 task sets. We conduct two sets of experiments:

1) We want to evaluate the performance of the two schedulability tests: BASIC and FAIR. In addition, we use the simple schedulability test from the stochastic capacity augmentation bound as baseline comparison.
2) We want to evaluate the different tardiness bounds of each individual task using different federated mapping algorithms. For task sets that are schedulable according the BASIC test, we record the maximum, mean and minimum tardiness of each task sets.

B. Experiment Results

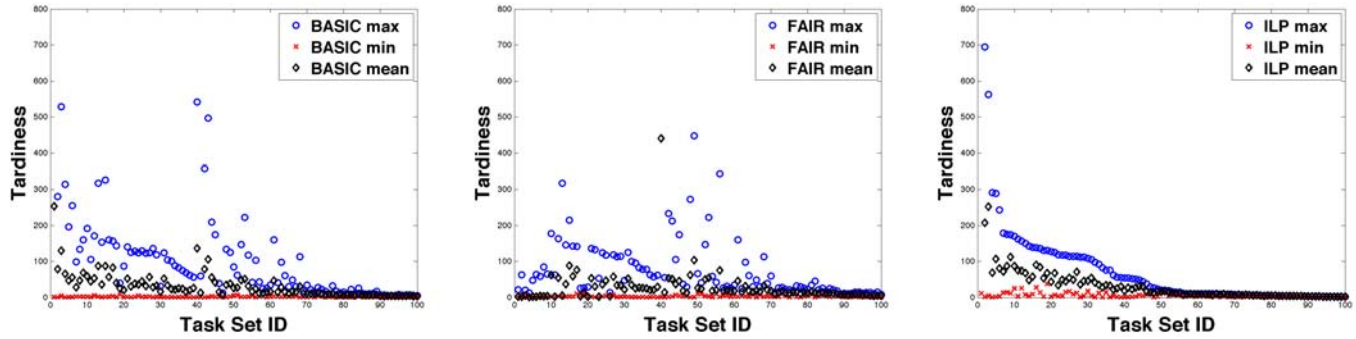
1) Schedulability Performance: We evaluate the performances of different schedulability tests: BOUND (as a baseline), BASIC and FAIR. Note that, as we have proved, the schedulability of ILP-based mapping algorithm is exactly the same with FAIR mapping algorithm. Also since the exact variance value of a task is not needed to run all these schedulability test, the schedulability performances of task sets with small variance and large variance are the same. Therefore, we do not include these curves in the figures.

From Figure 1, we can see that for all different numbers of cores, the FAIR/ILP algorithm performs the best, while BOUND performs the worst. Even though the bound indicates that task sets with total utilization larger than $50\%m$ may not be schedulable in terms of bounded tardiness, the two other linear time schedulability tests can still admits task sets up to around 60% for BASIC and 80% for FAIR.



(a) 8 cores (b) 16 cores (c) 32 cores

Fig. 1: Task Set Utilization vs. Schedulability Ratio (both in percentages) for different number of cores.



(a) BASIC mapping, small variance (b) FAIR mapping, small variance (c) ILP mapping, small variance

(d) BASIC mapping, large variance (e) FAIR mapping, large variance (f) ILP mapping, large variance

Fig. 2: Maximum, mean and minimum tardiness for parameters with small and large variances.

Also note that some task sets with 10% utilization are deemed unschedulable by BOUND. This is due to the critical-path length requirement for parallel tasks by BOUND. For a few tasks with 100% utilization, the FAIR algorithm still guarantees bounded tardiness, because all tasks in the set are low-utilization tasks. And GEDF scheduler can ensure bounded tardiness for sequential tasks with no utilization lost.

2) Tardiness of Tasks with Small and Large Variance: For task sets that bounded tardiness is guaranteed, we would like to compare the guaranteed expected tardiness. Note that both the LP and ILP optimization in FAIR and ILP mapping algorithms are only trying to optimize the maximum tardiness of the entire task sets. Therefore, it would be more interesting to see the different amount of expected tardiness bound for each individual task.

Figure 2 shows the maximum, mean and minimum expected tardiness calculated from the BASIC, FAIR and ILP mappings for task sets with small and large execution time variations respectively. To make it easy to compare, we sort all the figures according to the maximum tardiness of ILP mapping of that corresponding setting (low and high variances).

Not surprisingly, BASIC performs the worst among all three mappings, if we count the number of task sets for which BASIC generates the largest maximum tardiness. In fact, out of all randomly generated task sets, 92% and 85% task sets have smaller maximum tardiness by ILP than by BASIC, given small and large variance respectively. Compare FAIR and BASIC, 58% and 76% have lower maximum tardiness under FAIR.

However, we can also see that the maximum tardiness from BASIC mapping is comparable with (only slightly worse than) that from FAIR mapping, when variance of execution time and critical-path length is small. It is also comparable with ILP when variance is large. This is probably because all compared task sets satisfy the requirement of the bound. Therefore, there

is enough cores for BASIC mapping to approximate the better core assignment. Hence, when variation is small, one could use the BASIC mapping to bound the tardiness.

We also find that with large variance, the increase of maximum tardiness of FAIR is not significant, compared to that of BASIC and ILP. It is not surprising for BASIC result, because it confirms our hypothesis that the mapping of BASIC does not take into account of variation when allocating cores. However, ILP does try to balance the tardiness of all tasks considering variance similarly to FAIR.

In fact, comparing FAIR and ILP, we notice that 67% and 58% task sets have smaller maximum tardiness using ILP. ILP results seem worse with large variance, only because for some task sets, the maximum tardiness is from low-utilization tasks, which can be quite large. Hence, even through ILP can minimize the tardiness for high-utilization tasks, the LP calculation for low-utilization cannot directly minimize tardiness. As FAIR inflates the parameters for low-utilization tasks, the LP calculation may result in a better PS rate allocation and hence smaller tardiness.

IX. Conclusions

This paper analyze the soft real-time performance of federated scheduling for parallel real-time tasks of stochastic task models. This strategy provides the stochastic capacity augmentation bound of 2 for stochastic tasks with a soft real-time constraint of bounded expected tardiness. This is the such first result on stochastic parallel tasks. The federated scheduling strategy is promising due to its simplicity since it separately schedules high-utilization tasks on dedicated cores and low-utilization cores on shared cores; therefore, one can potentially use out-of-the-box schedulers in a prototype implementation.

References

[1] J. Kim, H. Kim, K. Lakashmanan, and R. Rajkumar, "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car," in *ICCPS '13*.

[2] J. Li, A. Saifullah, K. Agrawal, C. Gill, and C. Lu, "Capacity augmentation bound of federated scheduling for parallel dag tasks," Tech. Rep. WUCSE-2014-44, Washington University in St Louis, USA, 2014.

[3] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM*, vol. 46, no. 5, 1999.

[4] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," (Santa Barbara, California), pp. 207–216, July 1995.

[5] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comp. Surv.*, vol. 43, pp. 35:1–44, 2011.

[6] M. Bertogna and S. Baruah, "Tests for global edf schedulability analysis," *J. Syst. Archit.*, vol. 57, no. 5, pp. 487–497, 2011.

[7] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%," in *ECRTS '03*, 2003.

[8] J. M. López, J. L. Díaz, and D. F. García, "Utilization bounds for edf scheduling on real-time multiprocessor systems," *Real-Time Syst.*, vol. 28, pp. 39–68, Oct. 2004.

[9] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Improved multiprocessor global schedulability analysis," *Real-Time Syst.*, vol. 46, no. 1, pp. 3–24, 2010.

[10] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "A constant-approximate feasibility test for multiprocessor real-time scheduling," *Algorithmica*, vol. 62, no. 3-4, pp. 1034–1049, 2012.

[11] W. Y. Lee and H. Lee, "Optimal scheduling for real-time parallel tasks," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 6, pp. 1962–1966, 2006.

[12] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Inf. Process. Lett.*, vol. 106, no. 5, 2008.

[13] G. Manimaran, C. S. R. Murthy, and K. Ramamritham, "A new approach for scheduling of parallelizable tasks in real-time multiprocessor systems," *Real-Time Syst.*, vol. 15, no. 1, pp. 39–60, 1998.

[14] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS '09*.

[15] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS '10*.

[16] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *RTSS '11*.

[17] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques optimizing the number of processors to schedule multi-threaded tasks," in *ECRTS '12*.

[18] H. S. Chwa, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin, "Global edf schedulability analysis for synchronous parallel tasks on multicore platforms," in *ECRTS '13*, 2013.

[19] B. Andersson and D. de Niz, "Analyzing global-edf for multiprocessor scheduling of parallel tasks," in *Principles of Distributed Systems*, 2012.

[20] C. Liu and J. Anderson, "Supporting soft real-time parallel applications on multicore processors," in *RTCSA '12*.

[21] L. Nogueira and L. M. Pinho, "Server-based scheduling of parallel real-time tasks," in *International Conference on Embedded Software*, 2012.

[22] S. Baruah, V. Bonifaciy, A. Marchetti-Spaccamelaz, L. Stougiex, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *RTSS '12*.

[23] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic dag task model," in *ECRTS '13*.

[24] J. Li, K. Agrawal, C.Lu, and C. Gill, "Analysis of global edf for parallel tasks," in *ECRTS '13*.

[25] U. C. Devi, *Soft real-time scheduling on multiprocessors*. PhD thesis, University of North Carolina, 2006.

[26] A. Srinivasan and J. H. Anderson, "Efficient scheduling of soft real-time applications on multiprocessors.," in *ECRTS*, vol. 3, pp. 51–54, 2003.

[27] U. C. Devi and J. H. Anderson, "Tardiness bounds under global edf scheduling on a multiprocessor," *Real-Time Systems*, vol. 38, no. 2, pp. 133–189, 2008.

[28] J. Erickson, U. Devi, and S. Baruah, "Improved tardiness bounds for global edf," in *ECRTS '2010*.

[29] H. Leontyev and J. H. Anderson, "Generalized tardiness bounds for global multiprocessor scheduling," *Real-Time Systems*, vol. 44, no. 1-3, pp. 26–71, 2010.

[30] J. P. Erickson and J. H. Anderson, "Fair lateness scheduling: Reducing maximum lateness in g-edf-like scheduling,"

[31] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "An analytical bound for probabilistic deadlines," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pp. 179–188, IEEE, 2012.

[32] A. F. Mills and J. H. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *RTAS '10*, IEEE, 2010.

[33] J. M. López, J. L. Díaz, J. Entrialgo, and D. García, "Stochastic analysis of real-time systems under preemptive priority-driven scheduling," *Real-Time Systems*, vol. 40, no. 2, pp. 180–207, 2008.

[34] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *RTSS '02*, IEEE, 2002.

[35] "Intel CilkPlus." http://software.intel.com/en-us/articles/intel-cilk-plus.

[36] "OpenMP Application Program Interface v3.1," July 2011. http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

[37] J. Reinders, *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, 2010.

[38] K. Agrawal, C. E. Leiserson, Y. He, and W. J. Hsu, "Adaptive work-stealing with parallelism feedback," *ACM Trans. Comput. Syst.*, vol. 26, pp. 112–120, September 2008.

[39] J. Kingman, "Inequalities in the theory of queues," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 102–110, 1970.

[40] M. Tchiboukdjian, N. Gast, D. Trystram, J.-L. Roch, and J. Bernard, "A tighter analysis of work stealing," in *Algorithms and Computation*, pp. 291–302, Springer, 2010.

[41] O. Tardieu, H. Wang, and H. Lin, "A work-stealing scheduler for x10's task parallelism with suspension," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pp. 267–276, 2012.

[42] D. Leijen, W. Schulte, and S. Burckhardt, "The design of a task parallel library," in *Acm Sigplan Notices*, vol. 44, pp. 227–242, ACM, 2009.