Tiny Tasks – A Remedy for Synchronization Constraints in Multi-Server Systems

Markus Fidler   Brenton Walker   Stefan Bora
Institute of Communications Technology, Leibniz Universität Hannover

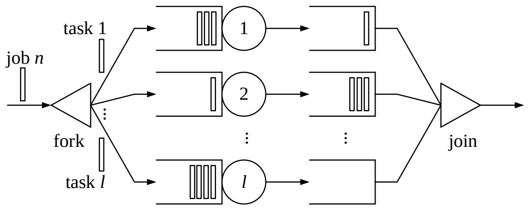


Fig. 1. Fork-join model.

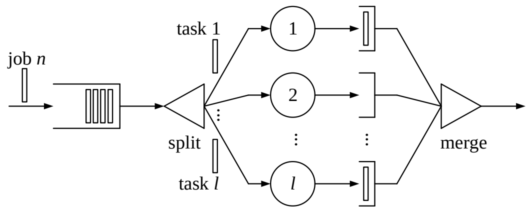


Fig. 2. Split-merge model.

***Abstract*—Models of parallel processing systems typically assume that one has l servers and jobs are split into an equal number of $k = l$ tasks. This seemingly simple approximation has surprisingly large consequences for the resulting stability and performance bounds. In reality, best practices for modern map-reduce systems indicate that a job's partitioning factor should be much larger than the number of servers available, with some researchers going to far as to advocate for a "tiny tasks" regime, where jobs are split into over 10,000 tasks. In this paper we use recent advances in stochastic network calculus to fundamentally understand the effects of task granularity on parallel systems' scaling, stability, and performance. For the split-merge model, we show that when one allows for tiny tasks, the stability region is actually much better than had previously been concluded. For the single-queue fork-join model, we show that sojourn times quickly approach the optimal case when l "big tasks" are sub-divided into $k \gg l$ "tiny tasks". Our results are validated using extensive simulations, and the applicability of the models used is validated by experiments on an Apache Spark cluster.**

I. Introduction

Two well-known models of parallel computation are the fork-join and split-merge models. Fig. 1 shows a schematic of the fork-join model. Jobs enter the system and are divided into k tasks (fork) that are assigned one by one to $l = k$ servers. Once all k tasks of a job are serviced, the job leaves the system (join). The difficulty in analyzing fork-join systems arises from the synchronization constraint of the join operation, and an exact solution is only known for the M|M|1 case with $k = l = 2$ [1], [2]. For broader classes of systems, a variety of approximation techniques have been used [2]–[10]. More recently several researchers have used stochastic network calculus to derive performance bounds [11]–[14].

The split-merge model, also referred to as "blocking fork-join" in [12], has an additional synchronization constraint. The system is blocked until the current job finishes service, hence a new job starts service only when all servers are idle. A schematic of the split-merge model is shown in Fig. 2. The analysis of the split-merge model turns out to be much simpler because it behaves like a single-server system with service times given by the service time of the largest task of each job [3], [15], [16]. The problem with the conventional ($k = l$) split-merge model is that it becomes unstable for utilizations well below one, and it becomes unstable more quickly as the degree of parallelism increases, as shown in Fig. 3. This has led some researchers to discount the model as impractical [12].

This work was supported in part by the German Research Council (DFG) under Grant VaMoS (FI 1236/7-1).

A third model arises in practice. When jobs are submitted by a multi-threaded driver program, map-reduce engines such as Apache Spark and Hadoop MapReduce do not typically behave like a traditional fork-join system, but rather as a single-queue fork-join system, where all tasks are held in a single FIFO queue and assigned to servers as they become available [17]. Compared to the fork-join model, where tasks are bound to particular servers and a large task can block tasks of subsequent jobs, in the single-queue fork-join model small jobs can overtake jobs with large straggler tasks. Mean sojourn times for such systems are derived in [18], and bounds on the sojourn time are derived using network calculus in [14].

Fig. 4 shows how job sojourn time scales with the number of servers for these three models given exponential inter-arrival and task service times. The plot shows performance bounds derived using network calculus in [12] and [14], simulation results, and experimental results from an Apache Spark cluster. For comparison the plot includes the equivalent sojourn time statistics for the ideal job partition. This is achieved when the jobs are partitioned into $k = l$ equally sized tasks. Both fork-join systems show a logarithmic increase in sojourn time as the degree of parallelism increases because of the synchronization constraint [10], [12]. The performance of the split-merge system appears catastrophic by comparison.

Apache Spark is a popular parallel processing engine that implements a map-reduce API [19], [20]. Fig. 4 shows that depending on the constraints put on the system, a Spark program

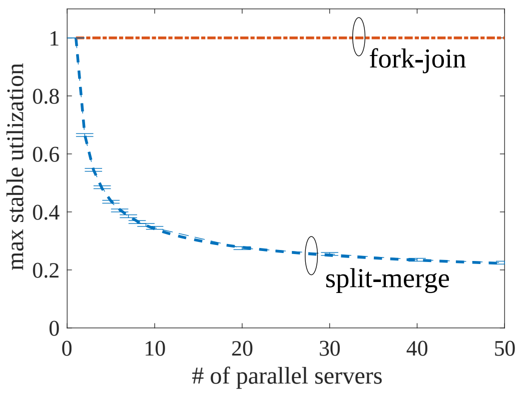


Fig. 3. Stability regions of the fork-join and split-merge models with l parallel servers for Exp(λ) inter-arrival times and Exp(μ) task service times, $\mu = 1$. The utilization is defined as $\varrho = \lambda/\mu$. Throughout this paper we generally plot analytical results with thick lines and simulation results with thin lines. We see that the fork-join system is stable up to a utilization of one regardless of the parallelism, but that the split-merge system quickly becomes unstable as the number of servers increases.

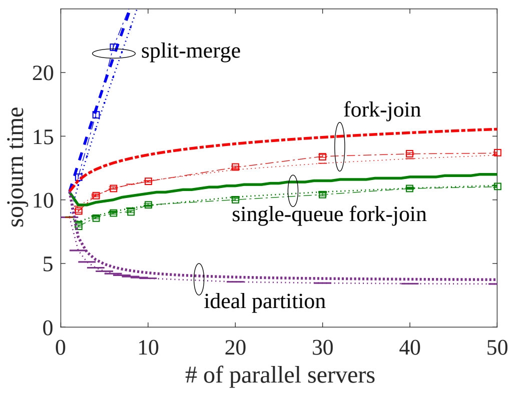


Fig. 4. Sojourn times of the conventional split-merge, fork-join, and single-queue fork-join models for varying degrees of parallelism. Job inter-arrival times are Exp($\lambda = 0.2$) and task service times are Exp($\mu = 1$). The 10^{-3} quantile of sojourn time is plotted. In addition to analytical bounds and simulation results, this shows experimental results from a Spark cluster marked with squares. We also include results for a parallel processing system with an ideal job partition, where a job is partitioned into l equally sized tasks. In case of the ideal partition all three models perform identically.

may exhibit the scaling behavior of split-merge, fork-join, or single-queue fork-join. When jobs are submitted from a single thread, as can easily be the case when jobs are submitted from an external script or interactive notebook, or when a busy programmer does not want to write multi-threaded code, the system scales like split-merge. When jobs are submitted from a multi-threaded driver program, the scheduler behaves like single-queue fork-join. If the system is configured with an extremely large `spark.locality.wait` value, then the performance will tend to scale like a fork-join system.

It is no surprise that map-reduce practitioners have devised methods to increase the performance of their systems, even in the split-merge case. The simplest of these is to partition jobs into a much larger number of tasks than there are servers, $k > l$. A common guideline is that the number of tasks should be about three times the number of servers, i.e., $k \approx 3l$ [21], or optimized through trial and error [22]. Some researchers have proposed even more extreme task granularity, coining the term "tiny tasks" [23]. So far, however, no analytical results studying how task granularity affects parallel system performance have been reported.

In this paper we will derive the stability region of the split-merge model with tiny tasks and quantify the improvement in the stability region compared to the equivalent "big tasks" model. In the limit, the stability region approaches one. Then we derive statistical performance bounds for both the split-merge and fork-join models with tiny tasks, and show how their performance improves over the equivalent big tasks model. In the limit the performance approaches that of the ideal job partition. Our results are validated by extensive simulations.

The remainder of this paper is structured as follows. We provide the required background on stochastic network calculus and discuss the state of the art in parallel systems in Sec. II. In Sec. III we prove how tiny tasks resolve the stability issues of the split-merge model. In Sec. IV we consider the fork-join model with tiny tasks and show that with increasing task tinyfication the performance of both the fork-join and the split-merge model approach that of the ideal job partition. We provide brief conclusions in Sec. V.

II. State of the Art

Here we present the analytical tools and notation needed to derive and understand our results. We build on the approach of [12], [14], making use of a max-plus version [24]–[27] of the stochastic network calculus [25], [28]–[33].

A. Notation and Background

Let $A(n)$ for $n \geq 1$ denote the **arrival time** of job n where $A(n) \geq A(m) \geq 0$ for $n \geq m \geq 1$. By convention, we use $A(0) = 0$. The **inter-arrival time** between job n and job m for $n \geq m \geq 1$ is $A(m,n) = A(n) - A(m)$. We denote the **departure time**, $D(n)$, using the same conventions. Servers are modeled using a definition of **max-plus server** with **service process** $S(m,n)$ that is adapted from [25, Def. 6.3.1].

Definition 1 (Max-plus server)**.** *A system with arrivals $A(n)$ and departures $D(n)$ is an $S(m,n)$ server under the max-plus algebra if it holds for all $n \geq 1$ that*

$$D(n) \leq \max_{m\in[1,n]} \{A(m) + S(m,n)\}.$$

Applying this definition to the **sojourn time**, $T(n) = D(n) - A(n)$ for $n \geq 1$, we obtain that

$$T(n) \leq \max_{m\in[1,n]} \{S(m,n) - A(m,n)\}. \quad (1)$$

In the case of first-come first-served service, the **waiting time** $W(n) = [D(n-1) - A(n)]_+$, where $[X]_+ = \max\{0, X\}$, can be derived in the same way.

For single-server systems the service process specifies the cumulative service time of jobs m to n and we have the relationship $S(m,n) = \sum_{\nu=m}^{n} \Delta(\nu)$ where $\Delta(\nu)$ is the service time of job ν. When we move to the multi-server setting, the definition of $S(m,n)$ becomes more subtle.

The derivations of waiting time and sojourn time bounds in [12], [14] make use of **moment generating functions** (MGFs) of the arrival process and the service process. The MGF of a random variable X is defined as $\mathsf{M}_X(\theta) = \mathsf{E}[e^{\theta X}]$ where θ is a free parameter. The MGF of a sum of independent random variables X and Y is the product of their individual MGFs, i.e., $\mathsf{M}_{X+Y}(\theta) = \mathsf{M}_X(\theta)\mathsf{M}_Y(\theta)$, and scaling X by a constant c is equivalent to a corresponding scaling of the parameter θ, i.e., $\mathsf{M}_{cX}(\theta) = \mathsf{M}_X(c\theta)$. A common class of MGF models are (σ, ρ)**-envelopes** defined in [25, Def. 7.2.1]. These are adapted to max-plus servers in [14, Def. 2].

Definition 2 ((σ,ρ)-Arrival and Service Envelopes)**.** *An arrival process, $A(m,n)$, is (σ_A, ρ_A)-lower constrained if for all $n \geq m \geq 1$ and $\theta > 0$ it holds that*

$$\mathsf{E}\left[e^{-\theta A(m,n)}\right] \leq e^{-\theta(\rho_A(-\theta)(n-m)-\sigma_A(-\theta))}.$$

A service process, $S(m,n)$, is (σ_S, ρ_S)-upper constrained if for all $n \geq m \geq 1$ and $\theta > 0$ it holds that

$$\mathsf{E}\left[e^{\theta S(m,n)}\right] \leq e^{\theta(\rho_S(\theta)(n-m+1)+\sigma_S(\theta))}.$$

(σ,ρ)-envelopes are models of G|G|1 queues, and a variety of stochastic processes satisfy the definition including Markov and periodic processes [25], [34]. In this work we restrict ourselves to GI|GI|1 queues with independent and identically distributed (iid) increments. For the arrival process $A(m,n) = \sum_{\nu=m}^{n-1} A(\nu,\nu+1)$ this means we assume iid inter-arrival times $A(\nu,\nu+1)$, and for the service process $S(m,n)$ the individual service times of each job $\Delta(n)$ are iid. In the iid case we have $\sigma_A(-\theta) = \sigma_S(\theta) = 0$.

As an example, consider the classical M|M|1 queue. The arrival process has iid inter-arrival times $A(n,n+1) \sim \text{Exp}(\lambda)$, and MGF $\mathsf{E}[e^{-\theta A(n,n+1)}] = \lambda/(\lambda+\theta)$ for $n \geq 1$ and $\theta > 0$. It follows that

$$\rho_A(-\theta) = -\frac{1}{\theta}\ln\left(\frac{\lambda}{\lambda+\theta}\right), \tag{2}$$

for $\theta > 0$. Similarly, for iid service times $\Delta(n) \sim \text{Exp}(\mu)$ we have $\mathsf{E}[e^{\theta\Delta(n)}] = \mu/(\mu-\theta)$ for $n \geq 1$ and $\theta \in (0,\mu)$ so that

$$\rho_S(\theta) = \frac{1}{\theta}\ln\left(\frac{\mu}{\mu-\theta}\right), \tag{3}$$

for $\theta \in (0,\mu)$. Parameter $\rho_A(-\theta)$ decreases with $\theta > 0$ from the mean inter-arrival time to the minimal inter-arrival time (possibly zero) and $\rho_S(\theta)$ increases with $\theta > 0$ from the mean service time to the maximal service time (possibly infinity).

Performance bounds are obtained using a basic theorem of the stochastic network calculus.

Theorem 1 (Statistical waiting and sojourn time bounds)**.** *Given an $S(m,n)$ server with iid inter-arrival times with envelope rate $\rho_A(-\theta)$ and iid service times with envelope rate $\rho_S(\theta)$. For any $\theta > 0$ that satisfies $\rho_S(\theta) \leq \rho_A(-\theta)$, the waiting time for all $n \geq 1$ is bounded by*

$$\mathsf{P}[W(n) > \tau] \leq e^{-\theta\tau},$$

and the sojourn time by

$$\mathsf{P}[T(n) > \tau] \leq e^{\theta\rho_S(\theta)}e^{-\theta\tau}.$$

Th. 1 is taken from [14, Th. 1] and can be found in a similar way, e.g., in [12], [13], [31], [35]–[37]. The waiting time bound also matches a classical result in [38]. Corresponding results are also available for the backlog and for the case of non-iid arrival and service processes. The free parameter θ can be optimized. For the example of the M|M|1 queue the maximal speed of the tail decay θ follows from $\rho_S(\theta) \leq \rho_A(\theta)$ as $\theta = \mu - \lambda$ so that $\mathsf{P}[T(n) > \tau] \leq \frac{\mu}{\lambda}e^{-(\mu-\lambda)\tau}$ [12, Eq. 14].

B. State of the Art in Parallel Systems

Here we will summarize prior results for split-merge, fork-join, single-queue fork-join, and ideal partitioning parallel systems. These are derived for the case where the number of tasks per job k equals the number of servers l. In the parallel setting we need to define additional terms and notation to distinguish between the service processes for jobs and tasks.

Given k tasks per job, let $Q_i(n)$ denote the **task service time** of task $i \in [1,k]$ of job $n \geq 1$. The **workload** of a job $L(n) = \sum_{i=1}^{k} Q_i(n)$ is defined to be the total of the service required by all of its tasks. The **job service time** $\Delta(n)$ is the total time a job spends in service. That is, the time between when its first task begins service and when all of its tasks finish service. Note that for the parallel models, $L(n)$ and $\Delta(n)$ are not necessarily equal and $S(m,n)$ may not generally be defined in increments of $\Delta(n)$.

1) Split-merge: In the split-merge model all $k = l$ tasks in a job start simultaneously. Therefore the service time of a job is determined by that of its maximal task $\Delta(n) = \max_{i\in[1,l]}\{Q_i(\nu)\}$. Hence, the model can be expressed as a max-plus server with service process

$$S(m,n) = \sum_{\nu=m}^{n} \max_{i\in[1,l]}\{Q_i(\nu)\}, \tag{4}$$

for $n \geq m \geq 1$ [12].

For the split-merge model with iid exponential task service times, the authors of [12] use the identity

$$\max_{i\in[1,l]}\{Q_i(\nu)\} =_d \sum_{i=1}^{l} \frac{Q_i(\nu)}{i}, \tag{5}$$

where $=_d$ denotes equality in distribution. It follows that for iid task service times $Q_i(n) \sim \text{Exp}(\mu)$, the mean job service time is $\mathsf{E}[\Delta(\nu)] = \sum_{i=1}^{l} 1/(i\mu)$. For iid inter-arrival times $A(n,n+1) \sim \text{Exp}(\lambda)$, i.e., with mean inter-arrival time $1/\lambda$, the split-merge system is stable if and only if [12, Eq. 21]

$$\frac{1}{\lambda} > \frac{1}{\mu}\sum_{i=1}^{l}\frac{1}{i}.$$

Then $\varrho = \lambda/\mu$ is the utilization. The term $\sum_{i=1}^{l} \frac{1}{i}$ is the lth harmonic number. These have the logarithmic asymptotic limit $\gamma + \ln l$, where $\gamma \approx 0.577$ is the Euler constant. Hence, the maximum stable utilization decays proportionally to $1/\ln l$ as seen in Fig. 3.

For iid $Q_i(n) \sim \text{Exp}(\mu)$ it also follows that the service process of the split-merge model (4) has service envelope

$$\rho_S(\theta) = \frac{1}{\theta} \sum_{i=1}^{l} \ln\left(\frac{i\mu}{i\mu - \theta}\right), \tag{6}$$

for $\theta \in (0, \mu)$. The sojourn time bound depicted in Fig. 4 is obtained by substitution of (6) into Th. 1.

2) Fork-join: Using the one-to-one mapping of $k = l$ tasks to l servers, the service process of the fork-join model is

$$S(m,n) = \max_{i\in[1,l]} \left\{ \sum_{\nu=m}^{n} Q_i(\nu) \right\}, \tag{7}$$

for $n \geq m \geq 1$ [12]. This says that $S(m,n)$ is determined by the maximal sequence of tasks that are assigned to a server. Clearly, for a given set of task service times $Q_i(n)$, the service process $S(m,n)$ of the fork-join model (7) will be less than or equal to that of the split-merge model (4). The sojourn time can be obtained from (1) by substitution of (7), $Q_i(m,n) = \sum_{\nu=m}^{n} Q_i(\nu)$, and reordering of the maxima to give

$$T(n) \leq \max_{i\in[1,l]} \left\{ \max_{m\in[1,n]} \{Q_i(m,n) - A(m,n)\} \right\}.$$

Then $T_i(n) = \max_{m\in[1,n]}\{Q_i(m,n) - A(m,n)\}$ are the individual task sojourn times at server $i \in [1,l]$. For each server $i \in [1,l]$ Th. 1 can be used to derive $\mathsf{P}[T_i(n) > \tau]$ and applying the union bound, [12], [14] gives us $\mathsf{P}[T(n) > \tau] \leq \sum_{i=1}^{l} \mathsf{P}[T_i(n) > \tau]$. The same steps can be used to derive a waiting time bound. For the homogeneous case it follows from Th. 1 that

$$\mathsf{P}[T(n) > \tau] \leq l e^{\theta\rho_Q(\theta)} e^{-\theta\tau},$$

for any $\theta > 0$ satisfying $\rho_Q(\theta) \leq \rho_A(-\theta)$. For the case of iid exponential arrivals and task service times, we can use $\rho_A(-\theta)$ from (2) and substitute $\rho_S(\theta)$ from (3) for $\rho_Q(\theta)$ to obtain the fork-join sojourn time bound plotted in Fig. 4.

Since $\rho_Q(\theta)$ and $\rho_A(-\theta)$ converge towards the mean task service time and the mean inter-arrival time, respectively, as $\theta \to 0$, the condition $\rho_Q(\theta) \leq \rho_A(-\theta)$ in Th. 1 implies that the model is stable up to a utilization of one, as seen in Fig. 3.

3) Single-queue fork-join: The service process of the single-queue fork-join model is more involved. The corresponding results in Fig. 4 are obtained from [14, Th. 4]. The single-queue fork-join model is also a special case (for $k = l$) of Th. 2 in this paper.

4) Ideal partition: If jobs are composed of l iid exponential tasks with parameter μ, then the jobs' total workload has distribution $L(n) \sim \text{Erlang}(l, \mu)$. If jobs with this workload distribution were instead divided into l equally-sized tasks, then the tasks would have an $\text{Erlang}(l, l\mu)$ distribution, so that

$$\rho_Q(\theta) = \frac{l}{\theta} \ln\left(\frac{l\mu}{l\mu - \theta}\right), \tag{8}$$

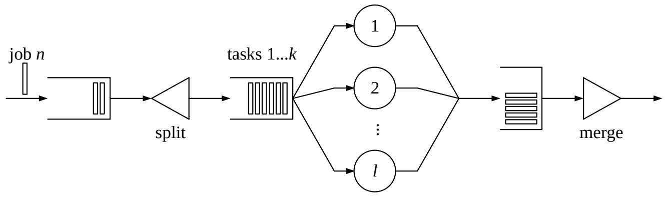


Fig. 5. Split-merge model with tiny tasks.

for $\theta \in (0, l\mu)$. Since the tasks of each job are equisized, all tasks of each job start and finish in unison. Hence, the system functions identically to a single server. The sojourn time bound depicted in Fig. 4 follows by substitution of (8) into Th. 1.

III. Stabilizing the Split-Merge System

In this section, we extend the split-merge model to cases with high task granularity, and show how tiny tasks extend its stability region and improve its sojourn time. In this model there are l servers and jobs are partitioned into $k \geq l$ tasks, where k may be orders of magnitude larger than l. We define $\kappa = k/l$ as the **factor of tinyfication**. The least granular case, where $\kappa = 1$ and $l = k$, is the conventional split-merge model. We will refer to the tasks in this case as "big tasks". When $\kappa > 1$ we refer to them as "tiny tasks".

A schematic of the split-merge tiny tasks model is shown in Fig. 5. Jobs are stored in a job queue, and if there is no job in service, the head-of-line job is partitioned into k tasks (split) which are then stored in the task queue. Since all servers are idle at the start of a job, the first l tasks start service immediately. Whenever a server finishes a task, it fetches the head-of-line task from the task queue. When all k tasks have finished service, the job leaves the system (merge) and the next job, if any, is partitioned and starts service.

When κ is an integer, we can directly compare the performance and stability of the big and tiny task systems. Conceptually we like to think of tiny tasks as being a refinement of the big tasks, but analytically it works best if we start with $k = \kappa l$ tiny tasks and aggregate them into l big tasks. If we have tiny task service times Q_i^{tiny}, then the corresponding big task service times would be $Q_j^{big} = \sum_{i=(j-1)\kappa+1}^{j\kappa} Q_i^{tiny}$. As this is just a re-grouping of the tasks, the job's workload distribution $L(n)$ is the same in both cases, and we can directly compare the effects of refining the granularity of the task partition. One distribution where this works well is when $Q_i^{tiny} \sim \text{Exp}(\mu)$. Then $Q_i^{big} \sim \text{Erlang}(\kappa, \mu)$, and the job workload has distribution $L(n) \sim \text{Erlang}(\kappa l, \mu)$ [39].

Lemma 1 (Tiny tasks split-merge model)**.** *The split-merge model with l servers and $k \geq l$ iid exponential tiny tasks with parameter μ is a max-plus server. Its service process has iid increments with envelope rate $\rho_S(\theta) = \rho_X(\theta) + (k-l)\rho_Z(\theta)$, where*

$$\rho_X(\theta) = \frac{1}{\theta} \sum_{i=1}^{l} \ln\left(\frac{i\mu}{i\mu - \theta}\right),$$

for $\theta \in (0, \mu)$, and

$$\rho_Z(\theta) = \frac{1}{\theta} \ln \left(\frac{l\mu}{l\mu - \theta} \right),$$

for $\theta \in (0, l\mu)$. The expected job service time is

$$\mathsf{E}[\Delta(n)] = \frac{1}{\mu} \left(\frac{k}{l} + \sum_{i=2}^{l} \frac{1}{i} \right).$$

We note that for the special case $k = l$, Lem.1 recovers the envelope rate (6) and stability condition of the conventional split-merge model. Sojourn time and waiting time bounds follow by substitution of Lem. 1 into Th. 1.

Proof. First, we show that the tiny tasks split-merge model is a max-plus server. Let $V_i(n)$ be the time task $i \in [1, k]$ of job $n \geq 1$ starts service. Since the first l tasks of a job start at the same time, we have for $i \in [1, l]$ that

$$V_i(n) = \max\{A(n), D(n-1)\}. \tag{9}$$

For $i \in [l+1, k]$ we have

$$V_i(n) = V_{i-1}(n) + Z_{i-1}(n), \tag{10}$$

where $Z_{i-1}(n)$ is the time from the start of task $i-1$ of job n until the next server becomes available.

We can express the departure time $D(n)$ of job n relative to the start time of its last task,

$$D(n) = V_k(n) + X(n), \tag{11}$$

where

$$X(n) = \max_{i \in [1,l]} \{Y_i(n)\} \tag{12}$$

and $Y_i(n)$ for $i \in [1, l]$ are the residual service times of the tasks, including task k, that are in service when task k starts service at $V_k(n)$. By repeated substitution of (10) into (11), it follows that

$$D(n) = V_l(n) + \left[\sum_{i=l}^{k-1} Z_i(n) \right] + X(n).$$

With (9) this becomes

$$D(n) = \max\{A(n), D(n-1)\} + \Delta(n), \tag{13}$$

where we write the service time of job n as

$$\Delta(n) = \left[\sum_{i=l}^{k-1} Z_i(n) \right] + X(n). \tag{14}$$

By recursive insertion of (13) we obtain

$$D(n) = \max_{m \in [1,n]} \left\{ A(m) + \sum_{\nu=m}^{n} \Delta(\nu) \right\},$$

i.e., the tiny tasks split-merge model is a max-plus server with service process $S(m, n) = \sum_{\nu=m}^{n} \Delta(\nu)$ and iid $\Delta(\nu)$.

Next, we consider the distribution of $X(n)$ and $Z_i(n)$. Due to the memorylessness of the iid exponential task service times, the residual service times $Y_i(n)$ are also iid exponential with the same parameter μ. Regarding $Z_i(n)$, note that when any task $i \in [l, k-1]$ of job n starts service all servers are busy, so that the time until the next server becomes idle is the minimum of the residual service times of the l tasks that are in service. Thus $Z_i(n)$ is the minimum of l iid exponential random variables with parameter μ, and therefore $Z_i(n) \sim \text{Exp}(l\mu)$.

To derive the MGF of (14), we apply (5) to (12), i.e., $X(n) = \max_{i \in [1,l]}\{Y_i(n)\} =_d \sum_{i=1}^{l} Y_i(n)/i$ which has MGF

$$\mathsf{M}[X(n)](\theta) = \prod_{i=1}^{l} \mathsf{M}\left[\frac{Y_i(n)}{i} \right](\theta) = \prod_{i=1}^{l} \frac{i\mu}{i\mu - \theta}, \tag{15}$$

for $\theta \in (0, \mu)$. Also, we have

$$\mathsf{M}\left[\sum_{i=l}^{k-1} Z_i(n) \right](\theta) = \mathsf{M}[Z_i(n)]^{k-l} = \left(\frac{l\mu}{l\mu - \theta} \right)^{k-l}, \tag{16}$$

for $\theta \in (0, l\mu)$. The MGF of (14) follows as the product of (15) and (16). Taking the logarithm and dividing by θ gives $\rho_S(\theta)$.

Finally, the expected value $\mathsf{E}[\Delta(n)]$ can then be derived by substituting (12) into (14) and using the identity (5). With $\mathsf{E}[Z_i(n)] = 1/(l\mu)$, and $\mathsf{E}[Y_i(n)/i] = 1/(i\mu)$ this gives us

$$\mathsf{E}[\Delta(n)] = \frac{k-l}{l\mu} + \frac{1}{\mu} \sum_{i=1}^{l} \frac{1}{i}.$$

Some reordering of the terms completes the proof. $\square$

A. Stability region of the tiny tasks split-merge model

The tiny tasks split-merge model is stable as long as the expected inter-arrival time $\mathsf{E}[A(n, n+1)]$ is larger than the expected job service time $\mathsf{E}[\Delta(n)]$. For iid inter-arrival times $A(n, n+1) \sim \text{Exp}(\lambda)$, the condition $\lambda \mathsf{E}[\Delta(n)] < 1$ implies stability. Since the expected total workload of a job is $\mathsf{E}[L(n)] = \sum_{i=1}^{k} \mathsf{E}[Q_i(n)] = k\mathsf{E}[Q_i(n)]$, the mean service provided to each job by each of the l servers will be $\kappa \mathsf{E}[Q_i(n)]$. The utilization of each server is then $\varrho = \lambda \kappa \mathsf{E}[Q_i(n)]$. Since $\lambda < 1/\mathsf{E}[\Delta(n)]$ for stability, the stability region, i.e., the maximum stable utilization for the tiny tasks model, is

$$\varrho < \frac{\kappa \mathsf{E}[Q_i(n)]}{\mathsf{E}[\Delta(n)]} = \frac{1}{1 + \frac{1}{\kappa} \sum_{i=2}^{l} \frac{1}{i}}, \qquad \text{(tiny tasks)} \tag{17}$$

where we inserted $\mathsf{E}[\Delta(n)]$ from Lem. 1 and $\mathsf{E}[Q_i(n)] = 1/\mu$ for $Q_i(n) \sim \text{Exp}(\mu)$ tiny tasks.

For comparison, consider the equivalent big tasks split-merge model where the number of tasks k equals the number of servers l and $Q_i(n) \sim \text{Erlang}(\kappa, \mu)$. The tiny tasks model above is a direct refinement of this model. From (4) the service process of the big tasks model is determined by the maximal task, $\Delta(n) = \max_{i \in [1,l]}\{Q_i(n)\}$. Since $\Delta(n)$ is non-negative, we can derive the expected value by integration of the complementary cumulative distribution function (CCDF) as

$$\begin{aligned} \mathsf{E}[\Delta(n)] &= \int_0^\infty 1 - \mathsf{P}[\Delta(n) \leq x] dx \\ &= \int_0^\infty 1 - (\mathsf{P}[Q_i(n) \leq x])^l dx, \end{aligned} \tag{18}$$

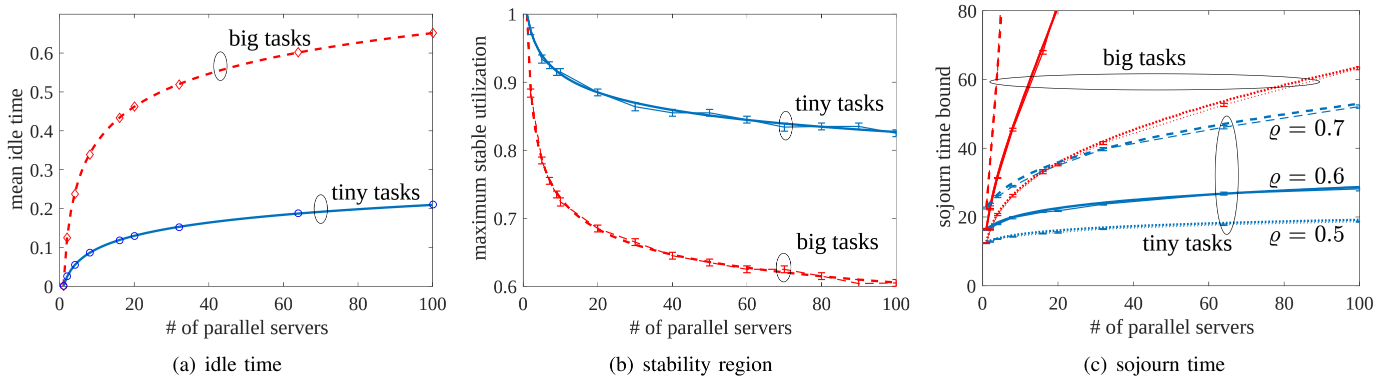


Fig. 6. Comparison of the split-merge model with tiny tasks vs. big tasks for varying number of servers, l, and Exp(λ) arrival process. Big tasks jobs have $k = l$ Erlang(κ, μ) tasks. Tiny tasks jobs have $k = \kappa l$ Exp(μ) tasks, and are therefore a direct refinement of the corresponding big tasks jobs. In all plots $\kappa = \mu = 20$ so the utilization is determined by the arrival rate $\varrho = \kappa\lambda/\mu = \lambda$. We see that using tiny tasks decreases the mean idle time (a) and increases the max stable utilization (b). The sojourn time statistics in (c) are computed for violation probability $\varepsilon = 10^{-6}$.

where we used that $\mathsf{P}[\max_{i\in[1,l]}\{Q_i(n)\} \le x] = \mathsf{P}[Q_1(n) \le x \cap Q_2(n) \le x \cap \dots \cap Q_l(n) \le x] = (\mathsf{P}[Q_i(n) \le x])^l$ for iid random variables $Q_i(n)$. Finally, we insert the Erlang-κ CDF

$$\mathsf{P}[Q_i(n) \le x] = 1 - e^{-\mu x} \sum_{i=0}^{\kappa-1} \frac{(\mu x)^i}{i!} \tag{19}$$

and solve (18) numerically. Again, $\lambda\mathsf{E}[\Delta(n)] < 1$ implies stability, and with $\varrho = \lambda\mathsf{E}[Q_i(n)]$, where $\mathsf{E}[Q_i(n)] = \kappa/\mu$ is the expected service time of the big tasks, the stability region for the big tasks model follows as

$$\varrho < \frac{\mathsf{E}[Q_i(n)]}{\mathsf{E}[\Delta(n)]} = \frac{\kappa}{\mu\mathsf{E}[\Delta(n)]} \qquad \text{(big tasks)} \tag{20}$$

where $\mathsf{E}[\Delta(n)]$ is given by (18).

The stability region of the split-merge model with both big tasks and tiny tasks for $\kappa = 20$ is shown in Fig. 6(b). The model with tiny tasks shows a clear improvement of the stability region.

1) Idle time: The differences in the stability region can be better understood by studying the idling of servers towards the end of a job. First we consider the idle times under the tiny tasks model with k iid Exp(μ) tasks. Let $V_k(n)$ denote the start of service of task k of job n. At this time l tasks are in service, and whenever a server finishes its task it idles until the job departs. The residual service times of these l tasks, $Y_i(n)$ for $i \in [1,l]$, are iid Exp(μ). We define the **idle time** of server i on job n, $I_i(n)$, to be the time between when server i finishes its processing for job n, and when job n departs. Note that all servers have the same idle time distribution, so we will simply write $I(n)$. The expected idle time for the tiny tasks model is then

$$\mathsf{E}[I(n)] = \mathsf{E}\left[\max_{i\in[1,l]}\{Y_i(n)\}\right] - \frac{1}{\mu} = \frac{1}{\mu}\sum_{i=2}^{l}\frac{1}{i}, \tag{21}$$

where we used (5). Note that this depends only on the number of servers l, not on the number of tasks per job k.

We note that by the linearity of the expectation, we can express the expected time a job spends in service $\mathsf{E}[\Delta(n)]$ in terms of the expected service time of the $\kappa = k/l$ tiny tasks that are assigned on average to each server plus the expected idle time, i.e., $\mathsf{E}[\Delta(n)] = \kappa\mathsf{E}[Q_i(n)] + \mathsf{E}[I(n)]$. This can also be verified by insertion of $\mathsf{E}[Q_i(n)] = 1/\mu$ and (21). The result matches $\mathsf{E}[\Delta(n)]$ in Lem. 1. Further, we can rewrite the stability region (17) by substitution of $\mathsf{E}[\Delta(n)] = \kappa\mathsf{E}[Q_i(n)] + \mathsf{E}[I(n)]$ as

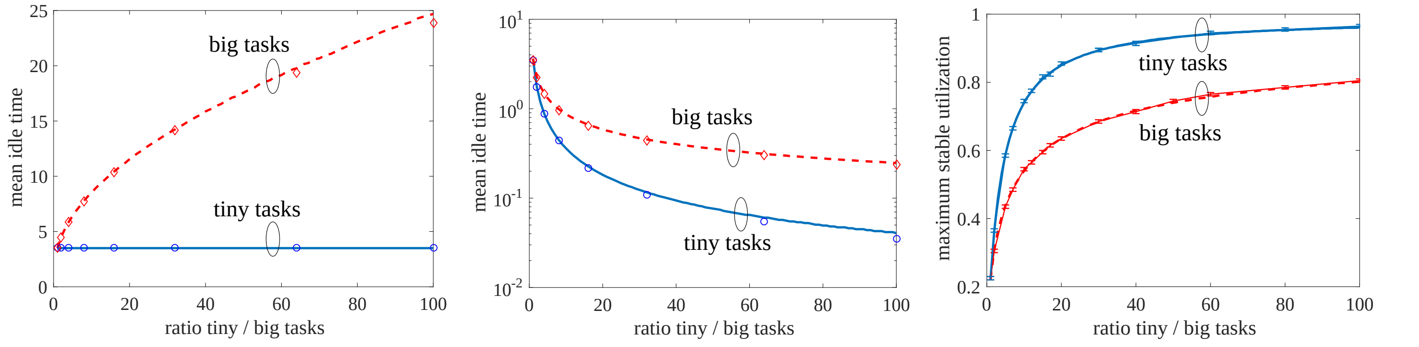
$$\varrho < \frac{1}{1 + \frac{\mu l}{k}\mathsf{E}[I(n)]}. \qquad \text{(tiny tasks)} \tag{22}$$

In the big tasks model with l Erlang-κ tasks, all l tasks start service at the same time, so the expected idle time is the difference between the expected service time of the maximal task $\mathsf{E}[\Delta(n)]$ given by (18) and the expected service time of any task $\mathsf{E}[Q_i(n)] = \kappa/\mu$. I.e., $\mathsf{E}[I(n)] = \mathsf{E}[\Delta(n)] - \mathsf{E}[Q_i(n)]$. Substitution into (20) matches (22) for the case $k = l$.

The expected idle times that correspond to the stability regions in Fig. 6(b) are shown in Fig. 6(a). For the parameters $\kappa = 20$ and $\mu = 20$, we have $\mathsf{E}[Q_i^{big}(n)] = \kappa\mathsf{E}[Q_i^{tiny}(n)] = 1$ so (22) simplifies to $\varrho < 1/(1 + \mathsf{E}[I(n)])$. For example, for $l = 100$ servers the mean idle time for the tiny tasks system is 0.21, giving a maximal stable utilization of 0.83. In the equivalent big tasks system, the mean idle time is 0.65, giving a maximal stable utilization of 0.60.

2) Scaling with l: The mean idle time of the tiny tasks model (21) is proportional to the lth harmonic number. Hence, the expected idle time grows logarithmically with the number of servers l, as shown in Fig. 6(a). It determines the reciprocal decrease of the stability region (22) we observe in Fig. 6(b).

3) Scaling with κ, k: We study the effects of increasing the degree of tinyfication, κ, from two different perspectives. First, in Fig. 7(a) we increase κ and hence k with a fixed mean task service time of $1/\mu$, for $\mu = 1$, $l = 50$. In the case of tiny tasks, idling is restricted to the last l tasks of each job. The mean idle time is given by (21) and does not change with k. Since the mean job size increases with k, however, the relative impact of the idle time diminishes and the stability region (22) improves. In case of big tasks, idling can occur



(a) idle time, constant μ (b) idle time, μ proportional to k (c) stability region

Fig. 7. Comparison of the split-merge model with tiny tasks vs. big tasks and varying tinyfication parameter κ, for $l = 50$ servers and Exp(λ) arrival process. Big tasks jobs are composed of l Erlang(κ, μ) tasks. Tiny tasks jobs are composed of $k = \kappa l$ Exp(μ) tasks. In 7(a) we hold $\mu = 1$ constant. In 7(b) we set $\mu = \kappa$ which holds the utilization constant. The maximum stable utilization for different values of κ is plotted in 7(c).

during the entire job. The mean idle time is computed from $\mathsf{E}[I(n)] = \mathsf{E}[\Delta(n)] - \mathsf{E}[Q_i(n)]$ which increases with κ.

Second, in Fig. 7(b) we increase κ, k, and μ proportionally, which holds the mean job workload constant $\mathsf{E}[L(n)] = k/\mu$. The decrease in mean idle time for the big tasks model is due to a reduction in the variability of the task sizes. While the mean size of a big task $\mathsf{E}[Q_i^{big}(n)] = \kappa/\mu$ stays constant, $\mathsf{Var}[Q_i^{big}(n)] = \kappa/\mu^2$ decreases. In case of tiny tasks, the mean size of the tasks decreases as their number increases, resulting in a significantly larger reduction of the idle times. In terms of equation (21), the mean idle time decreases because μ increases as the tasks get smaller.

The corresponding stability regions are shown in Fig. 7(c). The maximum stable utilization is completely determined by a given κ and l, so both idle time plots correspond to the same stability regions.

B. Sojourn time bounds

Fig. 6(c) compares sojourn time bounds of the big tasks and tiny tasks models for equivalent parameters. In the case of tiny tasks, the sojourn time bound is derived by substitution of parameter ρ_S from Lem. 1 into Th. 1. In case of big tasks, we first have to derive the envelope rate $\rho_S(\theta)$ of the service process $S(m,n)$ defined in (4) for iid tasks with $Q_i(n) \sim$ Erlang(κ, μ). We derive the MGF of $\Delta(n)$ by integration of the CCDF as

$$\mathsf{E}[e^{\theta\Delta(n)}] = \int_0^\infty 1 - \mathsf{P}[e^{\theta\Delta(n)} \leq x] dx.$$

Since for $\theta > 0$ it holds that $e^{\theta\Delta(n)} \geq 1$ we have

$$\mathsf{E}[e^{\theta\Delta(n)}] = 1 + \int_1^\infty 1 - \mathsf{P}\left[\Delta(n) \leq \frac{\ln(x)}{\theta}\right] dx.$$

By definition of $\Delta(n) = \max_{i\in[1,l]}\{Q_i(n)\}$ it follows that $\mathsf{P}[\Delta(n) \leq x] = (\mathsf{P}[Q_i(n) \leq x])^l$ so that

$$\mathsf{E}[e^{\theta\Delta(n)}] = 1 + \int_1^\infty 1 - \left(\mathsf{P}\left[Q_i(n) \leq \frac{\ln(x)}{\theta}\right]\right)^l dx.$$

We insert the Erlang-κ CDF (19) and solve the integral numerically. The envelope rate follows as $\rho_S(\theta) = \ln(\mathsf{E}[e^{\theta\Delta(n)}])/\theta$ and the sojourn time bound is derived by use of Th. 1.

The sojourn time bounds in Fig. 6(c) are shown for iid inter-arrival times $A(n, n+1) \sim$ Exp(λ). Three different λ are used, corresponding to utilizations of 0.5, 0.6, and 0.7. The use of tiny tasks improves the sojourn time bounds significantly. The improvement is larger under higher utilizations, where the big tasks split-merge model becomes unstable for even relatively small numbers of servers l.

IV. Approaching the Ideal Partition

We consider a single-queue fork-join model with tiny tasks. The model is similar to the split-merge model with tiny tasks depicted in Fig. 5, with one difference: there is no synchronization constraint at the start of a job. I.e., a new job can start service as soon as a server becomes idle and there are no unserviced tasks from the previous job. As a consequence, servers will not idle towards the end of a job if there are other jobs waiting. Furthermore, jobs can overtake each other and finish service out of sequence. We study a model where the jobs are forced to depart in sequence, i.e., $D(n) \leq D(n+1)$ for $n \geq 1$. That is, jobs that finish service must wait in a queue until their predecessors have departed.

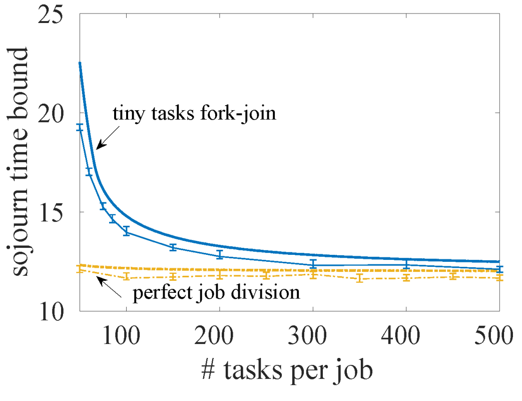
Using tiny tasks only makes sense in the context of a single-queue model. In the standard fork-join model, where tasks are bound to specific servers on arrival, tiny tasks would make no difference. Therefore, throughout this section we will refer to the single-queue model simply as fork-join with tiny tasks.

Theorem 2 (Tiny tasks fork-join model)**.** *Given a fork-join model with l servers and $k \geq l$ iid exponential tiny tasks with parameter μ and iid inter-arrival times with envelope rate $\rho_A(-\theta)$. For any $\theta \in (0, \mu)$ that satisfies $k\rho_Z(\theta) \leq \rho_A(-\theta)$, the waiting time of task $i \in [1, k]$ of job $n \geq 1$ is bounded by*

$$\mathsf{P}[W_i(n) \geq \tau] \leq e^{\theta(i-1)\rho_Z(\theta)} e^{-\theta\tau},$$

and the sojourn time of job $n \geq 1$ by

$$\mathsf{P}[T(n) \geq \tau] \leq e^{\theta((k-1)\rho_Z(\theta)+\rho_X)} e^{-\theta\tau}.$$



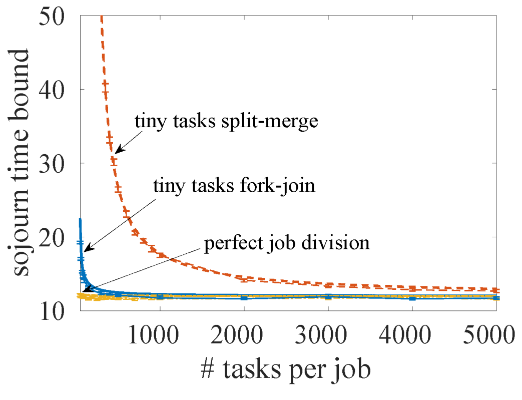


Fig. 8. Comparison of the sojourn time bounds of the single-queue fork-join and split-merge models with $l = 50$ servers and k tiny tasks. As a reference, the sojourn time bounds of a system with ideal partition, where a job is partitioned into l equisized tasks, is shown. Jobs have exponential inter-arrival times with parameter $\lambda = 0.5$ and are composed of k exponential tiny tasks with parameter $\mu = k$. The bounds are exceeded at most with $\varepsilon = 10^{-6}$.

The parameters $\rho_X(\theta)$ and $\rho_Z(\theta)$ are given in Lem. 1.

As a special case for $k = l = 1$, Th. 2 recovers the single server case Th. 1 for exponential jobs with envelope rate (3). Also, for $k = l$, Th. 2 recovers the waiting time bound for the single-queue fork-join model (without tiny tasks) [14, Th. 4]. For the sojourn time bound [14] uses a slightly different derivation technique that can provide tighter bounds mostly at low utilizations.

Proof. First, we derive a max-plus representation of the tiny tasks fork-join model. Let $V_i(n)$ be the time task $i \in [1, k]$ of job $n \geq 1$ starts service. For $i \in [2, k]$ and $n \geq 1$ we have

$$V_i(n) = V_{i-1}(n) + Z_{i-1}(n), \tag{23}$$

where $Z_{i-1}(n)$ is the time from the start of task $i-1$ of job n until the next server becomes idle and hence available. For $n = 1$ we have $V_1(1) = A(1)$ and for $i = 1$ and $n \geq 2$

$$V_1(n) = \max\{A(n), V_k(n-1) + Z_k(n-1)\}. \tag{24}$$

By repeated substitution of (23) into (24), it follows for $i \in [1, k]$ and $n \geq 1$ that

$$V_i(n) = \max_{m\in[1,n]} \{A(m) + Z_{i-1}(m, n)\}, \tag{25}$$

where

$$Z_{i-1}(m, n) = \sum_{j=1}^{i-1} Z_j(n) + \sum_{\nu=m}^{n-1} \sum_{j=1}^{k} Z_j(\nu). \tag{26}$$

It is straightforward to express the departure time $D(n)$ of job n as the maximum of the departure times of all tasks $i \in [1, k]$ of job n, $D(n) = \max_{i\in[1,k]}\{V_i(n) + Q_i(n)\}$ where $Q_i(n)$ is the service time of task i of job n. An alternative emerges when considering that at the start of service of task k of job n there are exactly l tasks of jobs $m \in [1, n]$ in service. Hence, job n will depart no later than

$$D(n) \leq V_k(n) + X(n), \tag{27}$$

where $X(n) = \max_{i\in[1,l]}\{Y_i(n)\}$ as in (12) and $Y_i(n)$ for $i \in [1, l]$ are the residual service times of the tasks, including task k of job n that are in service when task k of job n starts service. Among these l tasks may be tasks of earlier jobs $m < n$ that may finish later than the tasks of job n. Since the maximum in $X(n)$ in (27) implies that all tasks of all jobs $m \in [1, n]$ have finished service, our results incorporate the additional synchronization constraint that jobs depart in sequence. Inserting (25) into (27) this expands to

$$D(n) \leq \max_{m\in[1,n]} \{A(m) + Z_{k-1}(m, n) + X(n)\}.$$

If we define the job service process to be

$$S(m, n) = Z_{k-1}(m, n) + X(n)$$

then we obtain $D(n) \leq \max_{m\in[1,n]}\{A(m) + S(m, n)\}$, hence the fork-join tiny tasks model is a max-plus server.

As in the proof of Lem. 1, we have $Y_i(n) \sim \text{Exp}(\mu)$ iid, and $Z_i(n) \sim \text{Exp}(l\mu)$ iid. The envelope rates ρ_X and ρ_Z are derived exactly as in (15) and (16), respectively.

The waiting time of task $i \in [1, k]$ of job $n \geq 1$ is defined as $W_i(n) = V_i(n) - A(n)$. With (25) and a variable substitution it follows that $W_i(n) = \max_{m\in[1,n]}\{Z_{i-1}(n-m+1, n) - A(n-m+1, n)\}$. Hence, for any $\theta > 0$ we can write $\mathsf{P}[W_i(n) > \tau] = \mathsf{P}[\max_{m\in[1,n]}\{e^{\theta U_i(m)}\} > e^{\theta\tau}]$ where

$$U_i(m) = e^{\theta(Z_{i-1}(n-m+1,n) - A(n-m+1,n))}. \tag{28}$$

Similarly, the sojourn time of job $n \geq 1$ is defined as $T(n) = D(n) - A(n)$. Substituting (27) it follows that $T(n) \leq \max_{m\in[1,n]}\{Z_{k-1}(n-m+1, n) - A(n-m+1, n)\} + X(n)$. For any $\theta > 0$ we have $\mathsf{P}[T(n) > \tau] = \mathsf{P}[\max_{m\in[1,n]}\{e^{\theta U(m)}\} > e^{\theta\tau}]$ where

$$U(m) = e^{\theta(Z_{k-1}(n-m+1,n) - A(n-m+1,n) + X(n))}. \tag{29}$$

Given iid inter-arrival times it follows for (28) and (29) that

$$U_i(m+1) = U_i(m) e^{\theta(\sum_{j=1}^{k} Z_j(n-m) - A(n-m, n-m+1))},$$

where we used (26) and $A(n-m,n) = \sum_{\nu=n-m}^{n-1} A(\nu,\nu+1)$. Using the independence of $A(n-m,n-m+1)$ and $Z_j(n-m)$ we can write the conditional expectation

$$\begin{aligned}&\mathsf{E}[U_i(m+1)|U_i(m),U_i(m-1),\ldots,U_i(1)]\\ =&U_i(m)\mathsf{E}\left[e^{\theta\sum_{j=1}^{k} Z_j(n-m)}\right]\mathsf{E}\left[e^{-\theta A(n-m,n-m+1)}\right].\end{aligned}$$

Next, we apply the envelope rates $\rho_A(-\theta)$ of $A(n-m,n-m+1)$ and $\rho_Z(\theta)$ of $Z_j(n-m)$. If $k\rho_Z(\theta) \le \rho_A(-\theta)$, then $\mathsf{E}\left[e^{\theta\sum_{j=1}^{k} Z_j(n-m)}\right]\mathsf{E}\left[e^{-\theta A(n-m,n-m+1)}\right] \le 1$ and

$$\mathsf{E}[U_i(m+1)|U_i(m),U_i(m-1),\ldots,U_i(1)] \le U_i(m),$$

i.e., $U_i(m)$ is a supermartingale. Applying Doob's inequality for submartingales [40, Th. 3.2] and the reformulation for supermartingales [36], [37] for non-negative $U(m)$ it holds for $m \ge 1$ that

$$x\mathsf{P}\left[\max_{m\in[1,n]}\{U_i(m)\} \ge x\right] \le \mathsf{E}[U_i(1)]. \tag{30}$$

For the waiting time, we derive with (28) and $A(n,n) = 0$ that

$$\mathsf{E}[U_i(1)] = \mathsf{E}\left[e^{\theta\sum_{j=1}^{i-1} Z_j(n)}\right] \le e^{\theta(i-1)\rho_Z(\theta)}.$$

In the case of the sojourn time, we use (29) to derive

$$\mathsf{E}[U(1)] = \mathsf{E}\left[e^{\theta(\sum_{j=1}^{k-1} Z_j(n)} + X(n))\right] \le e^{\theta((k-1)\rho_Z(\theta)+\rho_X(\theta))}.$$

By insertion into (30) and letting $x = e^{\theta\tau}$ we obtain the bounds in Th. 2. □

In Fig. 8, we compare sojourn time bounds obtained for the single-queue fork-join and split-merge models with $l = 50$ servers and a varying number k of tiny tasks per job. For comparison we also show the sojourn time bounds of an equivalent system with the ideal partition of jobs into l equisized tasks. The bounds in the figure are evaluated with violation probability $\varepsilon = 10^{-6}$. In this comparison we increase the number of tasks per job k, and decrease the mean size of a task proportionally, so that the mean job workload $\mathsf{E}[L(n)]$ remains constant. Specifically we use iid inter-arrival times $A(n,n+1) \sim \text{Exp}(\lambda = 0.5)$, and iid task service times $Q_i(n) \sim \text{Exp}(k/l)$ for $l = 50$. Therefore the jobs' workloads have a $L(n) \sim \text{Erlang}(k,k/l)$ distribution, and constant mean $\mathsf{E}[L(n)] = l$ as we increase k. The utilization follows as $\varrho = \lambda = 0.5$. In case of the ideal partition, for an Erlang(k,μ) process the corresponding envelope rate would be

$$\rho_Q(\theta) = \frac{k}{\theta}\ln\left(\frac{\mu}{\mu-\theta}\right)$$

for $\theta \in (0, l\mu)$. Substituting $\mu = l\,k/l = k$ we obtain $\rho_Q(\theta)$ that can be inserted into Th. 1.

As the number of tasks k increases, the sojourn time bound of the fork-join model with tiny tasks quickly approaches that of the ideal partition, as seen in Fig. 8(a). Fig 8(b) includes sojourn time bounds for the tiny tasks split-merge model for comparison. For small k, the divergence between the models is quite large. This is a consequence of the restricted stability region of the split-merge model. For large k, both models approach the ideal partition.

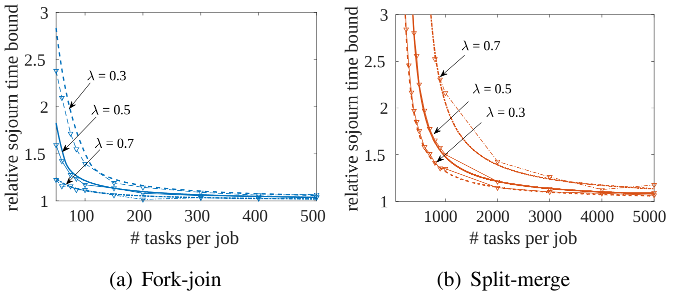


Fig. 9. Sojourn time bounds for the single-queue fork-join and split-merge models with tiny tasks, relative to the sojourn time bounds for the ideal partition for different utilizations. The remaining parameters are as in Fig. 8.

To see this convergence more directly, in Fig. 9 we plot the sojourn time statistics of the split-merge and fork-join systems relative to those achieved by the ideal task partition. This is shown for utilizations of 0.3, 0.5, and 0.7. The remaining parameters are as in Fig. 8. Here we see the relative sojourn times approach 1 as k becomes large. In the case of the fork-join model, shown in Fig. 9(a), the convergence is quicker if the utilization is larger. This is because the waiting time tends to dominate the sojourn time at higher utilization. Note that large straggler tasks (that are mitigated in case of tiny tasks) have a large impact on the time a job spends in service. However, due to flexible mapping of tasks to servers in the single-queue fork-join model, stragglers have little impact on the waiting time. This is different in case of the split-merge model where straggler tasks block the following jobs from entering service. A consequence of this is the observed reduction of the stability region due to idling of servers while waiting for stragglers. Due to the reduced stability region, the split-merge model has a much higher sensitivity to an increase of the utilization that can be mitigated only in case of a large degree of tinyfication.

V. Conclusions

We have shown, through analytical results and simulation, that increasing the granularity of the tasks in split-merge systems greatly increases their stability region. We similarly show that increasing task granularity improves the performance of both split-merge and fork-join systems, and that in the limit they approach the performance of the optimal task partition.

This work provides a theoretical basis for the common practice of addressing performance issues in parallel systems by increasing the granularity of the job partitioning. It provides a basis to understand the rate and limits of performance improvements possible through such refinements. Furthermore, this work challenges the conclusion that the split-merge model should be regarded as impractical. In fact, real world systems may behave like split-merge quite naturally, and the stability and performance issues are effectively mitigated through refinement of the task partitioning.

References

[1] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands," *SIAM J. Appl. Math.*, vol. 44, no. 5, pp. 1041–1053, 1984.

[2] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE Trans. Comput.*, vol. 37, no. 6, pp. 739–743, Jun. 1988.

[3] A. S. Lebrecht and W. J. Knottenbelt, "Response time approximations in fork-join queues," in *Proc. of UKPEW*, Jul. 2007.

[4] S.-S. Ko and R. F. Serfozo, "Sojourn times in G/M/1 fork-join networks," *Naval Research Logistics*, vol. 55, no. 5, pp. 432–443, May 2008.

[5] X. Tan and C. Knessl, "A fork-join queueing model: Diffusion approximation, integral representations and asymptotics," *Queueing Systems*, vol. 22, no. 3-5, pp. 287–322, Sep. 1996.

[6] S. Varma and A. M. Makowski, "Interpolation approximations for symmetric fork-join queues," *Performance Evaluation*, vol. 20, no. 1-3, pp. 245–265, May 1994.

[7] E. Varki, "Mean value technique for closed fork-join networks," *ACM Sigmetrics Perf. Eval. Rev.*, vol. 27, no. 1, pp. 103–112, May 1999.

[8] B. Kemper and M. Mandjes, "Mean sojourn times in two-queue fork-join systems: bounds and approximations," *OR Spektrum*, vol. 34, no. 3, pp. 723–742, Jul. 2012.

[9] F. Alomari and D. A. Menascé, "Efficient response time approximations for multiclass fork and join queues in open and closed queuing networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1437–1446, Jun. 2014.

[10] F. Baccelli, A. M. Makowski, and A. Shwartz, "The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds," *Adv. in Appl. Probab.*, vol. 21, no. 3, pp. 629–660, Sep. 1989.

[11] G. Kesidis, B. Urgaonkar, Y. Shan, S. Kamarava, and J. Liebeherr, "Network calculus for parallel processing," in *Proc. of MAMA Workshop at ACM SIGMETRICS*, Jun. 2015.

[12] A. Rizk, F. Poloczek, and F. Ciucu, "Stochastic bounds in fork-join queueing systems under full and partial mapping," *Queueing Systems: Theory and Applications*, vol. 83, no. 3, pp. 261–291, Aug. 2016.

[13] F. Poloczek and F. Ciucu, "Contrasting effects of replication in parallel systems: From overload to underload and back," in *Proc. of ACM SIGMETRICS*, Jun. 2016, pp. 375–376.

[14] M. Fidler, B. Walker, and Y. Jiang, "Non-asymptotic delay bounds for multi-server systems with synchronization constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1545–1559, Jul. 2018.

[15] P. Harrison and S. Zertal, "Queueing models with maxima of service times," in *Proc. of TOOLS*, Sep. 2003, pp. 152–168.

[16] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.

[17] B. Walker, "Benchmarking and simulating the fundamental scaling behaviors of a mapreduce engine," in *Proc. of ICFC*, Jun. 2017.

[18] R. Nelson, D. Towsley, and A. N. Tantawi, "Performance analysis of parallel processing systems," *IEEE Trans. Softw. Eng.*, vol. 14, no. 4, pp. 532–540, Apr. 1988.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. of USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[20] Apache Software Foundation, "Spark." [Online]. Available: https://spark.apache.org

[21] Apache Spark Documentation, *Tuning Spark*, 2019 (accessed 25 July 2019). [Online]. Available: https://spark.apache.org/docs/latest/tuning.html

[22] S. Ryza, *How-to: Tune Your Apache Spark Jobs (Part 2)*, 2015 (accessed 25 July 2019). [Online]. Available: https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

[23] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica, "The case for tiny tasks in compute clusters," in *HotOS'13*. USENIX, 2013.

[24] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley, 1992.

[25] C.-S. Chang, *Performance Guarantees in Communication Networks*. Springer-Verlag, 2000.

[26] J.-Y. Le Boudec and P. Thiran, *Network Calculus A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, 2001.

[27] J. Liebeherr, *Duality of the Max-Plus and Min-Plus Network Calculus*. Now Publishers, 2017.

[28] F. Ciucu, A. Burchard, and J. Liebeherr, "Scaling properties of statistical end-to-end bounds in the network calculus," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 2300–2312, Jun. 2006.

[29] M. Fidler, "An end-to-end probabilistic network calculus with moment generating functions," in *Proc. of IWQoS*, Jun. 2006, pp. 261–270.

[30] Y. Jiang and Y. Liu, *Stochastic Network Calculus*. Springer-Verlag, Sep. 2008.

[31] F. Ciucu and J. Schmitt, "Perspectives on network calculus - no free lunch but still good value," in *Proc. of ACM SIGCOMM*, Aug. 2012, pp. 311–322.

[32] M. Fidler, "A survey of deterministic and stochastic service curve models in the network calculus," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 1, pp. 59–86, 2010.

[33] M. Fidler and A. Rizk, "A guide to the stochastic network calculus," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 92–105, 2015.

[34] F. P. Kelly, "Notes on effective bandwidths," ser. Royal Statistical Society Lecture Notes. Oxford University, 1996, no. 4, pp. 141–168.

[35] F. Ciucu, "Network calculus delay bounds in queueing networks with exact solutions," in *Proc. of ITC-20*, Jun. 2007, pp. 495–506.

[36] Y. Jiang, "Network calculus and queueing theory: Two sides of one coin," in *Proc. of VALUETOOLS*, 2009, pp. 1–12, Invited Paper.

[37] ——, "A note on applying stochastic network calculus," Tech. Rep., 2010.

[38] J. F. C. Kingman, "A martingale inequality in the theory of queues," *Math. Proc. Cambridge*, vol. 60, no. 2, pp. 359–361, Apr. 1964.

[39] F. W. Steutel, "Random division of an interval*," *Statistica Neerlandica*, vol. 21, no. 34, pp. 231–244, 1967.

[40] J. L. Doob, *Stochastic Processes*. Wiley, 1953.