Generalized Parallel-Server Fork-Join Queues with Dynamic Task Scheduling

Mark S. Squillante
Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598, USA

Yanyong Zhang
Department of Electrical and Computer Engineering
Rutgers University
Piscataway, NJ 08854, USA

Anand Sivasubramaniam
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802, USA

Natarajan Gautam
Department of Industrial and Systems Engineering
Texas A&M University
College Station, TX 77843, USA

Abstract

This paper introduces a generalization of the classical parallel-server fork-join queueing system in which arriving customers fork into multiple tasks, every task is uniquely assigned to one of the set of single-server queues, and each task consists of multiple iterations of different stages of execution, including task vacations and communication among sibling tasks. Several classes of dynamic polices are considered for scheduling multiple tasks at each of the single-server queues to maintain effective server utilization. The paper presents an exact matrix-analytic analysis of generalized parallel-server fork-join queueing systems, for small instances of the stochastic model, and presents an approximate matrix-analytic analysis and fixed-point solution, for larger instances of the model.

Keywords: Quasi-birth-death processes, matrix-analytic methods, stochastic decomposition, parallel-server fork-join queues, stochastic networks.

1 Introduction

The classical parallel-server fork-join queueing system consists of P homogeneous servers each with their own infinite-capacity, first-come first-serve (FCFS) queue. A customer arrival is split into P tasks (fork) where the ith task is assigned to the ith single-server queue, $i \in \{1, \dots, P\}$. Every task consists of a service stage of execution at its assigned server followed by a synchronization stage of execution involving all sibling tasks (join). Each single-server queue serves its assigned tasks independent of the other queues. A customer departs the system after all of its tasks have completed the synchronization stage.

A wide variety of application domains have a similar fork-join structure and thus fork-join queueing systems have often played an important role in the modeling and analysis of these diverse applications; refer to, e.g., [21, 2, 30, 29, 6, 12] and the references therein. Over the past decade or so, however, many existing application domains have been evolving and many new application domains have been emerging, each exhibiting important differences in the properties of the tasks comprising every customer. Instead of having a single service stage followed by a single synchronization stage, the tasks of such applications are significantly more complex and they consist of different types of execution stages and multiple iterations of this collection of stages. In particular, a new class of execution stages is introduced in this paper to represent behavior where the task encounters any form of vacation, such as initiating an activity outside its assigned server and waiting for completion of this activity. The synchronization stage is replaced with a more general class of communication stages where the task sends and/or receives one or more messages to and/or from one or more sibling tasks. These different types of stages can be combined into any (relatively short) sequence of execution stages and the corresponding tasks are comprised of N iterations of this sequence, where N is a random variable.

While this generalized fork-join structure makes it possible to model the characteristics of emerging application domains, it also introduces additional complexities that do not exist in the classical parallel-server fork-join queueing system. In particular, the generalized fork-join structure can cause intervals of time where the task leaves idle the server to which it is assigned because of entering a vacation stage or waiting for a message within a communication stage. The FCFS discipline at every infinite-capacity single-server queue can then cause significant reductions in the stability region of the corresponding queueing system. We therefore allow the tasks of up to M customers to be allocated to the P single-server queues at any time. This in turn creates an opportunity for the dynamic scheduling of up to M tasks at each of the single-server queues, where a number of classes of basic dynamic task scheduling policies are of interest. These classes of scheduling policies cover a broad spectrum from little or no coordination among the single-server queues through various ways of providing coordination without any explicit synchronization among the single-server queues based on dynamic events at each queue.

The foregoing parallel-server fork-join queueing systems with dynamic task scheduling introduce fundamental extensions and new properties to the classical fork-join queue that are of general theoretical interest. In addition, some aspects of these extensions and new properties are motivated by a wide range of important application domains. This includes large-scale parallel computing environments (see, e.g., [10, 19, 5]) and various business applications such as the modeling and analysis of business processes (see, e.g., [9]). For example, recent trends from hundreds of computing servers to thousands and even tens of thousands of computing servers have caused major shifts in the design and implementation of both parallel computing applications and the policies used to schedule the execution of such applications. These computing environments are gaining acceptance not only in scientific application domains that require supercomputing power, but also in domains such as Web applications, Internet services, databases and multimedia, each of which place diverse workload demands on the underlying parallel system. The operations of these parallel applications further exhibit a broad spectrum of characteristics in terms of multiple iterations of computations followed by the reading and writing of data from and to memory or disk, followed by the sending and receiving of data to and from one or more sibling tasks, where the number of iterations can be fixed or based on a specified tolerance for the computed results. These diverse and complex parallel application characteristics raise complications in the scheduling of applications to achieve low response times together with high throughput and effective computing server utilizations. Hence, there also have been recent trends in the scheduling of parallel tasks away from the extremes of no coordination or completely-synchronized coordination among the computing servers and toward a class of dynamic scheduling policies, broadly called *dynamic coscheduling*; refer to, e.g., [7, 26, 17] and the references therein. This class of dynamic task scheduling policies attempts to approximate completely-synchronized coordination in large-scale parallel computing environments without any explicit synchronization among the computing servers by using local task events at each computing server (e.g., arrival of a communication message) to estimate the state of other computing servers and adjust the local scheduling of tasks accordingly.

We first present a formal definition of the proposed class of generalized parallel-server fork-join queueing systems with dynamic task scheduling, and then investigate a mathematical analysis of the inherent complexity of these queueing systems given the dynamic and complex interactions among the P single-server queues and the generalized fork-join customers. An exact matrix-analytic analysis [22, 23, 14] of the underlying stochastic processes is derived, which makes it possible to compute solutions for small P. We also derive an approximate analysis based on a general form of stochastic decomposition, a matrix-analytic analysis and a fixed-point iteration, which make it possible to compute solutions for large P. The results of our exact and approximate analysis provide an important characterization of the probabilistic and dynamic behavior of the multidimensional stochastic processes underlying the generalized parallel-server fork-join queueing systems with dynamic task scheduling. Furthermore, numerical experiments have been conducted to compare the results of our approximate solution against detailed simulations across a wide range of model parameter assumptions and settings. In each case, the results of our approximate analysis are in very good agreement with detailed simulations of a modest-size parallel system, often being within 5% and always less than 10%, where the accuracy of our approximate analysis increases as P grows. With a broad spectrum of new and emerging parallel computing environments comprised of thousands and even tens of thousands of computing servers, we are especially interested in a limiting regime based on generalized fork-join queueing systems with large numbers of parallel servers.

The remainder of the paper is organized as follows. Section 2 defines the proposed generalized parallel-server fork-join queueing systems with dynamic task scheduling. We then derive a general mathematical analysis of such parallel-server queueing systems in Section 3, followed by a brief discussion of applications of our results including a small representative sample of validation results. Section 5 provides concluding remarks, with the Appendix presenting an overview of some additional aspects of our analysis.

2 Model and Policy Descriptions

Consider a generalized parallel-server fork-join queueing system that consists of P homogeneous servers each with its own (local) queue of size $M < \infty$. Fork-join customers arrive to the queueing system from an exogenous source according to a Markovian Arrival Process (MAP) [23, 14] having descriptors $(\mathcal{S}_0^A, \mathcal{S}_1^A)$ of finite order m_A with mean rate $\lambda = (\mathbf{x}\mathcal{S}_1^A\mathbf{e})^{-1}$, where $\mathbf{e}$ denotes the column vector of appropriate dimension containing all ones and $\mathbf{x}$ is the invariant probability vector of the generator $\mathcal{S}^A = \mathcal{S}_0^A + \mathcal{S}_1^A$ (i.e., the unique solution of $\mathbf{x}\mathcal{S}^A = \mathbf{0}$ and $\mathbf{x}\mathbf{e} = 1$). An arriving fork-join customer is placed in a FCFS (global) system queue when the P single-server queues are already serving M parallel tasks. Upon an arriving customer finding less than M parallel tasks at the P single-server queues, the customer forks into P parallel tasks and the ith task is assigned to the ith single-server queue, $i \in \{1, \ldots, P\}$. Similarly, upon a customer departure when the P single-server queues are serving M parallel tasks, the customer at the head of the FCFS queue forks into P parallel tasks where the ith task is assigned to the ith single-server queue.

Remark 2.1. *A more general version of our generalized parallel-server fork-join queueing system consists of replacing the P single-server queues described above with P queues each having multiple homogeneous servers. This general model poses no fundamental difficulties in our mathematical analysis and can be handled in a straightforward manner, as noted in Section 3. However, we shall primarily focus on the model description above, since this general model further complicates the notation and exposition of our analysis without providing any significant new insights.*

Each parallel task consists of a (relatively short) sequence of execution stages that is repeated in an iterative manner. The number of iterations N for the parallel tasks of each fork-join customer is assumed to follow a (shifted) geometric distribution with parameter p_N, namely

$$\mathbb{P}[N = 1 + n] \ = \ (1 - p_N)^n p_N, \qquad\qquad n \in \mathbb{Z}_+.$$

We consider a sequence of execution stages that consists of a (standard) service stage, followed by a task vacation stage (where the task remains blocked until it returns from the vacation), followed by a communication stage (where the task sends and/or receives one or more messages to and/or from one or more sibling parallel tasks, with message sending operations preceding message receiving operations). The durations of these per-iteration service, vacation and communication stages are respectively assumed to be independent and identically distributed (i.i.d.) according to the finite order m^S, m^V and m^C phase-type distributions PH($\underline{\beta}, \mathcal{S}^S$), PH($\underline{\eta}, \mathcal{S}^V$) and PH($\underline{\zeta}, \mathcal{S}^C$) with means $\mu^{-1} = -\underline{\beta}(\mathcal{S}^S)^{-1}\mathbf{e}$, $\nu^{-1} = -\underline{\eta}(\mathcal{S}^V)^{-1}\mathbf{e}$ and $\gamma^{-1} = -\underline{\zeta}(\mathcal{S}^C)^{-1}\mathbf{e}$. (Here the superscripts S, V and C are intended to indicate that the parallel task is in a *service*, *vacation* or *communication* stage, respectively.) As part of any communication stage, a parallel task may also have to wait for the receipt of one or more messages from one or more sibling parallel tasks, including the classical fork-join synchronization where all parallel tasks of a fork-join customer must synchronize before proceeding. All of the above stochastic processes are assumed to be mutually independent.

While the above combination and sequence of execution stages comprising each parallel task of the queueing system are of general theoretical interest, some aspects of the model are also motivated by a broad spectrum of important applications; see, e.g., [10, 19, 5, 9] and the references cited therein. As a specific example, the computation of a parallel multidimensional FFT often requires each server to perform a one-dimensional FFT, following which the parallel servers exchange data with each other to implement a transpose, and the sequence of execution stages repeats in an iterative manner. Since the data sets are typically very large, the reading of data from the local disk at each server is often needed during the one-dimensional FFT. Even when one moves to newer parallel application domains such as video processing which requires high computational and data speeds to meet real-time requirements, each server waits for a video frame from another server, processes the frame, and then streams the result to the next server in a pipelined fashion. All of these application behaviors can be captured by the above generalized fork-join structure by using the appropriate setting of the model parameters together with the appropriate combinations and sequences of execution stages. It is also important to note that the mathematical analysis in Section 3 is not restricted to the specific generalized fork-join structure presented above, as our general approach can be extended in a relatively straightforward manner to support many different combinations and sequences of execution stages comprising the parallel task.

Each of the P single-server queues serves the up to M parallel tasks assigned to it independent of the other single-server queues. Several classes of basic policies are of interest for dynamically scheduling the parallel tasks at a single-server queue, each differing in the key decisions made to address various queueing system tradeoffs. Since the execution stages of parallel tasks can be highly variable with nonnegligible duration, we allow as a common attribute among these classes of dynamic scheduling policies the inclusion of a timesharing discipline in which the server is rotated in time among the set of parallel tasks (in their stages requiring the server) at the server. A penalty, or overhead, is incurred for switching from one task to another task at a single-server queue, independent of the cause. The lengths of the timesharing intervals and the overheads of switching between tasks at each single-server queue are respectively assumed to be i.i.d. following the phase-type distributions PH($\underline{\chi}, \mathcal{S}^Q$) and PH($\underline{\xi}, \mathcal{S}^O$) of finite orders m^Q and m^O having means $\tau^{-1} = -\underline{\chi}(\mathcal{S}^Q)^{-1}\mathbf{e}$ and $\delta^{-1} = -\underline{\xi}(\mathcal{S}^O)^{-1}\mathbf{e}$. (Here the superscripts Q and O are intended to indicate the corresponding timesharing *quantum* and task switching *overhead*.)

A server can become idle when the parallel task it is serving either enters a vacation stage or waits within a communication stage for one or more messages from one or more sibling parallel tasks. Conversely, the parallel task becomes available, or ready, for service due to either the completion of its vacation stage or the arrival of the message causing it to wait in a communication stage. The classes of basic scheduling policies of interest differ in the ways of handling these events and the corresponding complex dynamic interactions among the P single-server queues. While the diverse attributes of these classes of dynamic scheduling policies are of general theoretical interest, some aspects of the policies are also motivated by a wide variety of important applications; refer to, e.g., [10, 19, 5, 9] and the references therein. It is also important to note that the mathematical analysis in Section 3 is not restricted to these specific policy attributes and our approach can be extended in a relatively straightforward manner to support many different classes of dynamic scheduling policies at every single-server queue.

We now describe the classes of basic dynamic scheduling policies considered herein as part of our generalized parallel-server fork-join queueing system. In each case, there can be non-negligible overheads associated with the various actions and attributes of the policy, including the penalties for preempting and blocking tasks and the penalties for interrupting servers. Based on the known closure properties of phase-type distributions, these overheads can be captured in the model distributions above as well as in the mathematical analysis of Section 3.

No Coordination (NC). This policy schedules the parallel tasks at each single-server queue in a completely independent manner, thus providing no coordination among the P single-server queues. A priority list of parallel tasks available (or ready) for service is maintained, where the highest priority task is served for a timesharing quantum after which the task is placed at the end of the priority list and this priority dispatching procedure repeats. When a parallel task enters a vacation stage, it relinquishes the server and becomes blocked until this stage completes, at which point the parallel task is placed at the front of the priority list and immediately attains the server.

Pause and Block (PB). Under this policy, upon attempting to receive a message that has not arrived, the parallel task pauses for a period of time in the hope that the message will arrive within a small amount of time. The underlying rationale is that if the sender sibling parallel task is also currently in service at one of the other single-server queues, then the message is likely to arrive within a reasonably short period of time and, by having the (receiver) parallel task remain in service, this increases the likelihood of coordination among the sibling parallel tasks of a fork-join customer. Otherwise, if the message does not arrive within the pause period, then the parallel task becomes blocked until the message arrives so that server utilization is not wasted. Upon the arrival of a message when the corresponding (receiver) parallel task is blocked, the task is placed at the front of the priority list and immediately attains the server. Clearly, the length of the pause period and the magnitude of the various penalties incurred are important PB policy parameters that can impact the properties of the queueing system. There is an important tradeoff between wasting server utilization by making the pause period too long and interfering with the coordination among sibling parallel tasks by making the pause period too short. The same basic mechanism of pausing before blocking can be incorporated in any of the other classes of basic scheduling policies.

Event Driven (ED). This policy is similar to NC with the exception that the scheduling of parallel tasks is driven by the dynamic events encountered at the single-server queue. More specifically, the arrival of a message is used to place the parallel task for which the message is intended at the front of the priority list so that it immediately attains the server, thus preempting the parallel task currently in service if it is not the intended receiver. The rationale here is that the receipt of a message indicates the higher likelihood of the (sender) sibling parallel task currently being in execution at one of the other single-server queues. Clearly, the magnitude of the various penalties incurred are important ED policy parameters that can impact the properties of the queueing system.

Periodic Event Driven (PED). This policy is a less aggressive version of the ED policy that attempts to address potential inefficiencies arising from scheduling skews among the sibling parallel tasks of each fork-join customer. Instead of immediately interrupting the server upon the arrival of a message as in ED, the actions are somewhat delayed according to a procedure that periodically checks for a parallel task with a pending (i.e., arrived but not received) message. Upon finding such a task, the periodic procedure places this parallel task at the front of the priority list and it immediately attains the server. Even if there is no such task, but the currently scheduled parallel task is busy waiting for a message, the periodic procedure will place another parallel task with useful work to do at the front of the priority list so that it immediately attains the server. Clearly, the length of the period for running this procedure and the magnitude of the various penalties incurred are important PED policy parameters that can impact the properties of the queueing system. There is an important tradeoff between delaying task scheduling actions by making the periodic procedure run too infrequently and increasing overheads by making the periodic procedure run too frequently.

Finally, the properties of the stochastic processes assumed in our definition of the generalized parallel-server fork-join queueing system are important in that they determine both the generality of our solution and the usefulness of this mathematical model in practice. We therefore exploit the general class of MAPs and phase-type distributions; refer to [22, 23, 14] for more details including definitions, notation and known results. The use of MAPs and phase-type distributions for all model parameters is of theoretical importance in that we exploit their properties to derive solutions of various instances of our generalized parallel-server fork-join queueing system. It is also of practical importance in that, since the class of phase-type distributions is dense within the set of probability distributions on $[0, \infty)$, and since the class of MAPs provides a general framework for capturing the dependence structure and variability of a process, any stochastic process on this space for the parallel-server fork-join queueing systems of interest can in principle be represented arbitrarily closely by a MAP or a phase-type distribution. Moreover, a considerable body of research has examined the fitting of phase-type distributions and MAPs to empirical data, and a number of algorithms have been developed for doing so; see, e.g., [1, 8, 15, 25, 24] and the references cited therein.

3 Mathematical Analysis

In this section we present a mathematical analysis of the introduced class of generalized parallel-server fork-join queueing systems. Some of the difficulties in deriving such an analysis center around the inherent complexity of these queueing systems given the dynamic and complex interactions among the P single-server queues and the generalized fork-join customers. Hence, we first present a general formulation of the underlying stochastic processes and derive an exact matrix-analytic analysis that makes it possible to compute solutions for small P. To address some of the problems involved in computing exact solutions for large instances of this complex parallel-server queueing system, we derive an approximate analysis based on a general form of stochastic decomposition that provides probabilistic characterizations of the dynamic interactions and dependencies among the P single-server queues under each dynamic task scheduling policy. These probabilistic characterizations are then exploited to obtain a reduced state-space representation, for which we derive a matrix-analytic analysis based on a fixed-point iteration. Our approximate analysis can be shown to be asymptotically exact in terms of the number of servers P, under certain conditions, and simulation-based experiments demonstrate that our approach is very accurate for modest-size parallel-server systems and that this accuracy increases with P (see Section 4). Motivated by many different application domains, some involving thousands and even tens of thousands of servers, we are particularly interested in generalized fork-join queueing systems with large numbers of parallel servers. Lastly, we provide expressions for various performance measures in terms of our exact and approximate solutions, and then extend both solution approaches by incorporating a very simple first-order approximation of the impact of resource contention.

3.1 Exact Matrix-Analytic Analysis

The generalized parallel-server fork-join queueing system can be modeled by the continuous-time Markov process $\{X(t)\,;\, t \in \mathbb{R}_+\}$, defined on the multi-dimensional state space $\Omega^X = \bigcup_{i=0}^{\infty} \Omega_i^X$ where

$$\Omega_0^X \equiv \left\{ (0, \overline{j}^A) \mid \overline{j}^A \in \omega^A \right\}, \tag{1}$$

$$\Omega_i^X \equiv \left\{ (i, \overline{j}^A, \overline{j}_1^S, \ldots, \overline{j}_P^S) \mid \overline{j}^A \in \omega^A,\ \overline{j}_k^S \in \omega_k^S \right\}, \qquad i \in \mathbb{Z}^+, \tag{2}$$

$$\begin{aligned} \omega_k^S = \Big\{ & (i_k, \overline{j}_k^R, j_{k,1}^S, \ldots, j_{k,\min\{i_k,M\}}^S, j_k^Q) \mid i_k \in \{0, \ldots, \min\{i, M\}\}, \\ & \overline{j}_k^R(\ell) \in \{1, \ldots, \min\{i_k, M\}\},\ j_{k,\ell}^S \in \{1, \ldots, \sigma_{N_\phi}\}, \\ & j_k^Q \in \{1, \ldots, m^Q + m^O\},\ \ell \in \{1, \ldots, \min\{i_k, M\}\} \Big\}, \qquad k \in \{1, \ldots, P\}, \end{aligned} \tag{3}$$

$$\sigma_n \equiv \sum_{\ell=1}^{n} \phi_\ell, \tag{4}$$

and ω^A is the set of possible states for the MAP arrival process. The state-vector variable i denotes the total number of fork-join customers in the system; $\overline{j}^A$ denotes the state of the MAP arrival process; i_k denotes the number of parallel tasks assigned to server k; $\overline{j}_k^R(\ell)$ denotes the index of the ℓ^{th} priority ordered parallel task assigned to server k (where $\overline{j}_k^R(1)$ denotes the index of the parallel task currently using server k) when $i_k \geq 1$ and it is defined to be equal to 1 when $i_k = 0$; $j_{k,\ell}^S$ denotes the phase of the *overall service process* (defined below) for the ℓ^{th} task assigned to server k; j_k^Q denotes the phase of the timesharing quantum length process (including task switching overhead) at server k; ϕ_ℓ denotes the number of phases in the phase-type distribution for the ℓ^{th} execution stage of the overall service process; and N_ϕ denotes the number of execution stages comprising the overall service process. Note that our exact analysis can address the general model of Remark 2.1 in a straightforward manner by augmenting the set ω_k^S and the corresponding state transitions to capture the multiple parallel servers at each queue $k \in \{1, \ldots, P\}$.

The above formulation exploits known closure properties of phase-type distributions [22, 14], most notably that the convolution of phase-type distributions is also of phase-type. We therefore can generally capture the various stages of execution (e.g., service, vacation and communication) for the classes of generalized fork-join structures (and variants thereof) of interest via a single phase-type distribution that represents the appropriate combinations of the phase-type distributions for each of these stages of execution as well as other system behavior (e.g., the impact of a task waiting to receive a message). We refer to this combined process as the *overall service process*. Moreover, as described in Section 3.2, we note that one of the general keys of our approximate approach consists of capturing the various probabilistic and dynamic behaviors of the parallel-server fork-join queueing system, including its task scheduling policy, in the construction and use of this overall service process.

Let $x_{i,d} \in \Omega_i^X$, $d \in \{1, \ldots, D_i^X\}$, $i \in \mathbb{Z}_+$, be a lexicographic ordering of the elements of Ω_i^X, and define $D^X \equiv \sum_{i=0}^{M-1} D_i^X$, where D_i^X denotes the cardinality of Ω_i^X. The subset Ω_i^X is called *level* i of

$\{X(t)\,;\,t\in\mathbb{R}_+\}$, $i\in\mathbb{Z}_+$. Define

$$\begin{aligned}
\boldsymbol{\pi}^X &\equiv (\boldsymbol{\pi}_0^X, \boldsymbol{\pi}_1^X, \boldsymbol{\pi}_2^X, \ldots),\\
\boldsymbol{\pi}_i^X &\equiv (\pi^X(x_{i,1}), \pi^X(x_{i,2}), \ldots, \pi^X(x_{i,D_i^X})), \qquad i\in\mathbb{Z}_+,\\
\pi^X(x_{i,d}) &\equiv \lim_{t\to\infty}\mathbb{P}[X(t)=x_{i,d}], \qquad i\in\mathbb{Z}_+,\;\; x_{i,d}\in\Omega_i^X,\;\; d\in\{1,\ldots,D_i^X\}.
\end{aligned}$$

Then the limiting probability vector $\boldsymbol{\pi}^X$ is the stationary distribution for the Markov process $\{X(t)\,;\,t\in\mathbb{R}_+\}$. Assuming this process to be irreducible and positive recurrent, the invariant probability vector is uniquely determined by solving $\boldsymbol{\pi}^X\mathbf{Q}^X=\mathbf{0}$ and $\boldsymbol{\pi}^X\mathbf{e}=1$, where $\mathbf{Q}^X$ is the infinitesimal generator matrix for the process. The elements of the generator matrix are constructed based on the above formulation and the dynamic transitions among the stages of behavior for the generalized parallel-server fork-join queueing system, which also depend upon the specific dynamic task scheduling policy. An example of this construction is postponed until Section 3.2, since the transitions between states follow the same logic in both our exact and approximate analysis, with the exact analysis simply requiring the application of this logic to the entire set of single-server queues in all possible combinations.

The generator matrix $\mathbf{Q}^X$ has the form

$$\mathbf{Q}^X = \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots\\ \mathbf{B}_{10} & \mathbf{B}_{11} & \mathbf{A}_0 & \mathbf{0} & \mathbf{0} & \ldots\\ \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \mathbf{0} & \ldots\\ \mathbf{0} & \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \ldots\\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \tag{5}$$

where $\mathbf{B}_{00}$, $\mathbf{B}_{01}$, $\mathbf{B}_{10}$, $\mathbf{B}_{11}$ and $\mathbf{A}_n$, $n=0,1,2$, are finite matrices of dimensions $D^X\times D^X$, $D^X\times D_M^X$, $D_M^X\times D^X$, $D_M^X\times D_M^X$ and $D_M^X\times D_M^X$, respectively. Markov processes having this block-tridiagonal structure are called *quasi-birth-death* (QBD) processes because the transitions from each state are restricted to states in the same level or states in the two adjacent levels. The matrices corresponding to the non-homogeneous boundary of Ω^X have the form

$$\mathbf{B}_{00} = \begin{pmatrix} \boldsymbol{\Psi}_0 & \boldsymbol{\Lambda}_0 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0}\\ \boldsymbol{\Phi}_1 & \boldsymbol{\Psi}_1 & \boldsymbol{\Lambda}_1 & \mathbf{0} & \cdots & & \\ \mathbf{0} & \boldsymbol{\Phi}_2 & \boldsymbol{\Psi}_2 & \boldsymbol{\Lambda}_2 & \cdots & \vdots & \vdots\\ \vdots & \vdots & \vdots & \vdots & \ddots & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\Phi}_{M-1} & \boldsymbol{\Psi}_{M-1} \end{pmatrix}, \qquad \mathbf{B}_{01} = \begin{pmatrix} \mathbf{0}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\\ \boldsymbol{\Lambda}_{M-1} \end{pmatrix}, \tag{6}$$

$$\mathbf{B}_{10} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{\Phi}_M \end{pmatrix}, \qquad \mathbf{B}_{11} = \boldsymbol{\Psi}_M, \tag{7}$$

where $\boldsymbol{\Phi}_i$, $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Lambda}_i$ have dimensions $D_i^X\times D_{i-1}^X$, $D_i^X\times D_i^X$ and $D_i^X\times D_{i+1}^X$, respectively.

We know from standard matrix-analytic results that the stationary distribution $\boldsymbol{\pi}^X$ of the QBD process $\{X(t)\,;\,t\in\mathbb{R}_+\}$ with generator $\mathbf{Q}^X$ has a matrix-geometric form given by

$$\boldsymbol{\pi}_{M+k} = \boldsymbol{\pi}_M\mathbf{R}^k, \qquad k\in\mathbb{Z}_+, \tag{8}$$

$$(\boldsymbol{\pi}_0,\,\boldsymbol{\pi}_1,\,\ldots,\,\boldsymbol{\pi}_M)\begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01}\\ \mathbf{B}_{10} & \mathbf{B}_{11}+\mathbf{R}\mathbf{A}_2 \end{pmatrix} = \mathbf{0}, \tag{9}$$

$$(\boldsymbol{\pi}_0,\,\boldsymbol{\pi}_1,\,\ldots,\,\boldsymbol{\pi}_{M-1})\,\mathbf{e} + \boldsymbol{\pi}_M(\mathbf{I}-\mathbf{R})^{-1}\mathbf{e} = 1, \tag{10}$$

where $\mathbf{R}$ is the minimal nonnegative matrix that satisfies $\mathbf{R}^2\mathbf{A}_2 + \mathbf{R}\mathbf{A}_1 + \mathbf{A}_0 = \mathbf{0}$. Furthermore, the time and space complexities of computing (9) for large values of M can be significantly reduced by exploiting the following standard results. We first derive from (6), (7), (9) the equations

$$\boldsymbol{\pi}_k = -\boldsymbol{\pi}_{k+1}\,\boldsymbol{\Phi}_{k+1}\,\widetilde{\mathbf{R}}_k^{-1}, \qquad k \in \{0,\ldots,M-1\}, \tag{11}$$

$$\boldsymbol{\pi}_M = -\boldsymbol{\pi}_{M-1}\,\boldsymbol{\Lambda}_{M-1}\,\widetilde{\mathbf{R}}_M^{-1}, \tag{12}$$

where $\widetilde{\mathbf{R}}_0 = \boldsymbol{\Psi}_0$, $\widetilde{\mathbf{R}}_k = \boldsymbol{\Psi}_k - \boldsymbol{\Phi}_k\widetilde{\mathbf{R}}_{k-1}^{-1}\boldsymbol{\Lambda}_{k-1}$, for all $k \in \{1,\ldots,M-1\}$, $\widetilde{\mathbf{R}}_M = \boldsymbol{\Psi}_M + \mathbf{R}\mathbf{A}_2$. Then we can obtain the probability vector $\boldsymbol{\pi}_M$ up to a multiplicative constant by solving

$$\boldsymbol{\pi}_M\left[\boldsymbol{\Psi}_M + \mathbf{R}\mathbf{A}_2 - \boldsymbol{\Phi}_M\widetilde{\mathbf{R}}_{M-1}^{-1}\boldsymbol{\Lambda}_{M-1}\right] = \mathbf{0}, \tag{13}$$

$$\boldsymbol{\pi}_M\,\mathbf{e} = 1, \tag{14}$$

and we can recursively solve for the remaining probability vectors $\boldsymbol{\pi}_{M-1},\ldots,\boldsymbol{\pi}_0$ up to a multiplicative constant using (11). The final solution is obtained through the normalizing equation (10).

Remark 3.1. *Our analysis above considers the process $\{X(t)\,;\,t\in\mathbb{R}_+\}$ as a level-independent QBD process with a level-dependent boundary, where we exploit the matrix-analytic structure of the level-dependent boundary in equations* (11) – (14)*. Alternatively, we can instead consider the process $\{X(t)\,;\,t\in\mathbb{R}_+\}$ as a level-dependent QBD process, and thus exploit computationally efficient algorithms for calculating the stationary distribution $\boldsymbol{\pi}$; e.g., refer to [3, 4, 13, 14]. This is the preferred approach for such calculations. However, since our approximate analysis exploits the physical interpretation of various aspects of the generalized parallel-server fork-join queueing system, we shall primarily consider the underlying stochastic process as a level-independent QBD process with a level-dependent boundary in order to simplify the notation and presentation of our analysis.*

Using the foregoing approach makes it possible for us to compute exact solutions for small instances of the generalized parallel-server fork-join queueing systems, which are otherwise prohibitively expensive. Still, a primary difficulty in this approach for solving larger instances of $\{X(t)\,;\,t\in\mathbb{R}_+\}$ is the combinatorial explosion of the size and complexity of Ω^X, which records the state of each single-server queue and task in the system. We next derive our approximate solution approach that addresses these issues.

3.2 Stochastic Decomposition

A general approach in applied probability for solving multidimensional stochastic processes with various forms of correlation basically consists of decomposing the complex dependencies and dynamic interactions among the different dimensions into simpler stochastic processes with reduced dimensionality and then approximating these dependencies and dynamic interactions through a system of fixed-point equations; see, e.g., [27] and the references cited therein. In some cases, it can be shown that this fixed-point solution converges asymptotically to the exact solution in a (problem-specific) limiting regime. One classical example of this general approach is the well-known Erlang fixed-point approximation in which the multidimensional Erlang formula is approximated by a system of equations in terms of the one-dimensional Erlang formula, and which has been shown to be asymptotically exact in a limiting regime as the traffic intensities and resource capacities increase together in proportion; refer to, e.g., [11] and the references therein.

Our approximate analysis is based on a version of this general stochastic decomposition approach to create a tractable formulation for large instances of the complex multidimensional Markov process of Section 3.1. In particular, we approximate the generalized parallel-server fork-join queueing system by assuming that the distribution of the state of each single-server queue is stochastically independent and identical to the distribution of the states of the other single-server queues. The corresponding decomposed Markov process $\{Y(t)\,;\,t \in \mathbb{R}_+\}$, representing each individual single-server queue, then can be solved in isolation by modifying its overall service process to reflect the probabilistic and dynamic behaviors of the other single-server queues in the parallel-server fork-join queueing system. Hence, the complex dependencies among the single-server queues (and the parallel tasks assigned to them) and their complex dynamic interactions are probabilistically captured, in part, by exploiting the overall service process. The stationary distribution for the entire generalized parallel-server fork-join queueing system can be obtained by analyzing the single-server queue in this manner via a fixed-point iteration. Our approximate analysis can be shown to be asymptotically exact in the limit as $P \to \infty$ under certain assumptions, following the results in [20].

3.2.1 Matrix-Analytic Analysis

The particular single-server queue of interest can be modeled by the continuous-time Markov process $\{Y(t)\,;\,t \in \mathbb{R}_+\}$, defined on the multi-dimensional state space $\Omega^Y = \bigcup_{i=0}^{\infty} \Omega_i^Y$ where

$$\Omega_0^Y \equiv \left\{ (0, \overline{j}^A) \mid \overline{j}^A \in \omega^A \right\}, \tag{15}$$

$$\Omega_i^Y \equiv \left\{ (i, \overline{j}^A, \overline{j}^R, j_1^S, \ldots, j_{\min\{i,M\}}^S, j^Q) \mid \overline{j}^A \in \omega^A,\ \overline{j}^R(\ell) \in \{1, \ldots, \min\{i, M\}\}, \right.$$
$$\left. j_k^S \in \{1, \ldots, \sigma_{N_\phi}\},\ j^Q \in \{1, \ldots, m^Q + m^O\},\ k, \ell \in \{1, \ldots, \min\{i, M\}\} \right\}, \quad i \in \mathbb{Z}^+. \tag{16}$$

The state-vector variable i denotes the number of parallel tasks at the given single-server queue; $\overline{j}^A$ denotes the state of the MAP arrival process; $\overline{j}^R(\ell)$ denotes the current index of the ℓ^{th} priority ordered parallel task assigned to the single-server queue (where $\overline{j}^R(1)$ denotes the index of the parallel task currently using the server) when $i \geq 1$ and it is defined to be equal to 1 when $i = 0$; j_k^S denotes the phase of the overall service process for the k^{th} parallel task assigned to the server; and j^Q denotes the phase of the timesharing quantum length process (including task switching overhead) at the single-server queue. Recall that N_ϕ denotes the number of execution stages comprising the overall service process, that ϕ_ℓ denotes the number of phases in the phase-type distribution for the ℓ^{th} execution stage of the overall service process, $\ell \in \{1, \ldots, N_\phi\}$, and that ω^A and σ_n are as given in Section 3.1. Note that our approximate analysis can address the general model of Remark 2.1 in a straightforward manner by augmenting the set Ω_i^Y and the corresponding state transitions to capture the multiple parallel servers at each of the P queues.

Let $y_{i,d} \in \Omega_i^Y$, $d \in \{1, \ldots, D_i^Y\}$, $i \in \mathbb{Z}_+$, be a lexicographic ordering of the elements of Ω_i^Y, and define $D^Y \equiv \sum_{i=0}^{M-1} D_i^Y$, where D_i^Y denotes the cardinality of Ω_i^Y. The subset Ω_i^Y is called *level* i of $\{Y(t)\,;\,t \in \mathbb{R}_+\}$. Define the stationary probability vector $\boldsymbol{\pi}^Y$ for the Markov process $\{Y(t)\,;\,t \in \mathbb{R}_+\}$ in a manner exactly analogous to our definition of $\boldsymbol{\pi}^X$ in Section 3.1. Assuming this process to be irreducible and positive recurrent, the invariant probability vector is uniquely determined by solving $\boldsymbol{\pi}^Y \mathbf{Q}^Y = \mathbf{0}$ and $\boldsymbol{\pi}^Y \mathbf{e} = 1$, where $\mathbf{Q}^Y$ is the infinitesimal generator matrix for the process. The generator matrix $\mathbf{Q}^Y$ has the form of (5) through (7) where $\mathbf{B}_{00}$, $\mathbf{B}_{01}$, $\mathbf{B}_{10}$, $\mathbf{B}_{11}$, $\mathbf{A}_n$, $n = 0, 1, 2$, $\boldsymbol{\Phi}_i$, $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Lambda}_i$ are finite matrices of dimensions $D^Y \times D^Y$, $D^Y \times D_M^Y$, $D_M^Y \times D^Y$, $D_M^Y \times D_M^Y$, $D_M^Y \times D_M^Y$, $D_i^Y \times D_{i-1}^Y$, $D_i^Y \times D_i^Y$ and

$D_i^Y \times D_{i+1}^Y$, respectively. The invariant probability vector $\boldsymbol{\pi}^Y$ of the QBD process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$ with generator $\mathbf{Q}^Y$ then can be obtained from equations (8) through (14).

The elements of the generator matrix are constructed based on the above formulation and the dynamic transitions among the stages of behavior for the generalized parallel-server fork-join queueing system, which also depend upon the dynamic task scheduling policy. A key aspect of our stochastic decomposition approach consists of capturing in the overall service process these complex dependencies and dynamic interactions among the P single-server queues and the up to M generalized fork-join customers, and constructing the decomposed stochastic process of the single-server queue to include these probabilistic and dynamic behaviors. To elucidate the exposition of our approach, we consider a generic example consisting of M parallel tasks at each of the single-server queues under a simple variant of the NC policy that allocates the server in timesharing intervals to every task in a round-robin fashion. Additional details for each dynamic task scheduling policy are provided in the Appendix.

The first 3 stages of the overall service process for each parallel task represent the service, vacation and communication stages of execution, and thus

$$\phi_1 \;=\; m^S, \qquad\qquad \phi_2 \;=\; m^V, \qquad\qquad \phi_3 \;=\; m^C.$$

In the present example, we assume that the communication stage of each task consists of sending a *single* message to one other task followed by receiving a *single* message from one other task. (Note that more general cases of sending/receiving multiple messages are addressed within the context of our stochastic decomposition approach in Section 3.2.2.) We further assume that the communication stage is comprised of the total server demands for the send operation and the portion of the server demands for the receive operation up to the point of checking if the message from another task has arrived; the server demands for the remainder of the receive operation are included as part of the service stage of the next iteration. For this reason, we shall henceforth use "send stage" and "communication stage" interchangeably, unless noted otherwise. Recall that a task switch occurs on a vacation operation. At the end of the vacation stage for each iteration, the customer departs the system with probability p_{N_A}, and otherwise proceeds to the send stage. (Note that having the customer depart at the end of any other execution stage can be easily addressed by adjusting the details of our approach in an analogous manner.) Upon completion of this communication stage, the parallel task immediately enters the service stage of the next iteration if the message to be received has already arrived, which occurs with probability $\widehat{p}_A(\cdot)$. Note that if $\widehat{p}_A(\cdot) \equiv 1$, then the overall service process for each parallel task would consist solely of execution stages 1, 2 and 3, where the impact of timesharing quanta, task switch overheads and task scheduling dynamics in the service stages and communication stages (after each vacation stage) of the parallel task being modeled is directly supported in our analysis of these execution stages. On the other hand, if the message to be received has not arrived, then the sender of the message can be in 1 of 4 (generic) states, namely service, vacation, send and waiting for a message from yet another sibling parallel task. Note that the additional execution stages of the overall service process are used to capture these different states of behavior for the sender task and the resulting delays experienced by the parallel task being modeled. The corresponding events that the sender of the message is in the service, vacation or send stages of execution occur with probability $\widehat{p}_S(\cdot)$, $\widehat{p}_V(\cdot)$ and $\widehat{p}_C(\cdot)$, respectively.

Let T_S denote the maximum number of iterations, at any given time, that a waiting task can be ahead of the sibling parallel task that will be sending the corresponding message, under the dynamic task scheduling policy. To further clarify the presentation, suppose that $T_S = 0$ in our generic example which implies that the sender of a message to any task in a waiting stage must be in the same iteration (albeit in an earlier stage) as the waiting task. (Note that the general case for T_S is considered in Section 3.2.2.) We therefore have that the sender can be in its service stage, its vacation stage or its communication stage at the point in time when the modeled parallel task starts to wait upon attempting to receive the message. Hence,

$$N_\phi \;=\; 6, \qquad\qquad \phi_4 \;=\; m^{S'}, \qquad\qquad \phi_5 \;=\; m^{V'}, \qquad\qquad \phi_6 \;=\; m^{C'},$$

where the corresponding phase-type distributions for execution stages 4, 5 and 6 are denoted by PH($\underline{\beta}'_{(\cdot)}, \mathcal{S}^{S'}_{(\cdot)}$), PH($\underline{\eta}'_{(\cdot)}, \mathcal{S}^{V'}_{(\cdot)}$) and PH($\underline{\zeta}'_{(\cdot)}, \mathcal{S}^{C'}_{(\cdot)}$). These order ϕ_4, ϕ_5, ϕ_6 phase-type distributions and the measures $\widehat{p}_A(\cdot)$, $\widehat{p}_S(\cdot)$, $\widehat{p}_V(\cdot)$, $\widehat{p}_C(\cdot)$ are all calculated from our decomposed single-server queue solution and form the basis for a fixed-point iteration that is derived in Section 3.2.2. More general communication strategies are handled in a similar manner. For example, in the case of an all-to-all communication strategy (where all sibling parallel tasks of a fork-join customer send and receive a message from each other), we derive a formula for the distribution of time that the task waits for messages from all other tasks (based on distributions conditioned on the state of the system), and then we use a phase-type distribution to estimate (or bound) this waiting time distribution.

The corresponding set of transition rates for this generic example is obtained in a relatively straightforward manner based on the formulation and analysis described above. We note that the general case of these transition rules are similarly obtained in a relatively straightforward manner. Moreover, the corresponding set of general rules for other types of generalized fork-join structures (cf. Section 2) and dynamic task scheduling policies (cf. Section 2) can be constructed in a similar fashion within the context of our stochastic decomposition approach. In particular, NC is quite similar to the general case of the policy considered above, where the vector $\overline{j}^R(\ell)$ is used to provide a priority boost to each customer immediately upon the completion of its vacation operation. A similar approach is used to address PB, with the addition of extending the phase-type distribution for the communication stage to include pause period and interrupt costs (as part of the corresponding receive operation). Somewhat more complex priority orderings are maintained in the vector $\overline{j}^R(\ell)$ for the ED and DED policies in order to properly capture the local execution of parallel tasks under these schemes.

To simply illustrate the relatively straightforward procedure for constructing the transition rates of the generator matrix, we present in Table 1 the set of transition rules corresponding to a specific instance of the generic example discussed above. (Additional information on the construction of the state transitions is provided in the Appendix.) The MAP arrival process reduces to a phase-type renewal process with interarrival distribution PH($\underline{\alpha}, \mathcal{S}^A$). In addition to the constraints on the state variables defined in equations (15) and (16), we assume that $r \in \{1, \ldots, m_A\}$, $u_\ell \in \{1, \ldots, \phi_\ell\}$, $q \in \{1, \ldots, m_Q\}$ and $o \in \{1, \ldots, m_O\}$, unless noted otherwise. Let $\mathcal{Q}(j^Q)$ denote the constraint $j^Q \in \{1, \ldots, m^Q\}$, let $\mathcal{Q}'(j^Q)$ denote the constraint $j^Q \in \{m^Q + 1, \ldots, m^Q + m^O\}$, and let $\mathcal{R}_\ell(j^S_k)$ denote the constraint $j^S_k \in \{\sigma_{\ell-1} + 1, \ldots, \sigma_\ell\}$. For notational convenience, define $\mathbf{k} \equiv \overline{j}^R$, $\mathbf{f} \equiv \mathbf{k}(1)$ (i.e., the index of the executing task), $\mathbf{k}^+ \equiv [\mathbf{k}(2), \ldots, \mathbf{k}(1)]$, $\mathbf{k}^- \equiv [\ldots, \mathbf{k}(k-1), \mathbf{k}(k+1), \ldots]$, $\mathbf{i} \equiv \min\{i, M\}$, $\mathbf{v}_\ell \equiv \sigma_{\ell-1} + u_\ell$ and $\sigma_0 \equiv 0$. Since the objective is to simply illustrate our general approach, and in order to elucidate the exposition of these rules, we will further assume in the table that departures from level 1 are as defined for levels 2 through M but with the other components of the state vector appropriately removed, and that fork-join customers depart the system immediately upon completion of the final vacation operation (without gaining access to the server).

From $(i, j^A, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, j^Q)$ To State	Transition Rate
Arrivals:	
$(i, r, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, j^Q), \quad r \neq j^A$	$\mathcal{S}^A(j^A, r)$
$(i+1, r, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, \mathbf{v}_1, j^Q), \quad 0 \leq i \leq M-1$	$\underline{S}^A(j^A) \cdot \underline{\alpha}(r) \cdot \underline{\beta}(u_1)$
$(i+1, r, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, j^Q), \quad i \geq M$	$\underline{S}^A(j^A) \cdot \underline{\alpha}(r)$
Timesharing Quanta and Task Switches:	
$(i, j^A, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, q), \quad \mathcal{Q}(j^Q), q \neq j^Q, i \geq 1$	$\mathcal{S}^Q(j^Q, q)$
$(i, j^A, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, m^Q + o), \quad \mathcal{Q}(j^Q), i \geq 1$	$\underline{S}^Q(j^Q) \cdot \underline{\xi}(o)$
$(i, j^A, \mathbf{k}, j_1^S, \ldots, j_\mathbf{i}^S, m^Q + o), \quad \mathcal{Q}'(j^Q), m^Q + o \neq j^Q, i \geq 1$	$\mathcal{S}^O(j^Q - m^Q, o)$
$(i, j^A, \mathbf{k}^+, j_1^S, \ldots, j_\mathbf{i}^S, q), \quad \mathcal{Q}'(j^Q), i \geq 1$	$\underline{S}^O(j^Q - m^Q) \cdot \underline{\chi}(q)$
Execution Stage 1:	
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_1, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_1(j_\mathbf{f}^S), \mathbf{v}_1 \neq j_\mathbf{f}^S, i \geq 1$	$\mathcal{S}^S(j_\mathbf{f}^S, u_1)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_2, j_{\mathbf{f}+1}^S, \ldots, m^Q + o), \quad \mathcal{Q}(j^Q), \mathcal{R}_1(j_\mathbf{f}^S), i \geq 1$	$\underline{S}^S(j_\mathbf{f}^S) \cdot \underline{\eta}(u_2) \cdot \underline{\xi}(o)$
Execution Stage 2:	
$(i, j^A, \mathbf{k}, \ldots, j_{k-1}^S, \mathbf{v}_2, j_{k+1}^S, \ldots, j^Q), \quad \mathcal{R}_2(j_k^S), \mathbf{v}_2 \neq j_k^S, i \geq 1, 1 \leq k \leq \mathbf{i}$	$\mathcal{S}^V(j_k^S - \sigma_1, u_2)$
$(i, j^A, \mathbf{k}, \ldots, j_{k-1}^S, \mathbf{v}_3, j_{k+1}^S, \ldots, j^Q), \quad \mathcal{R}_2(j_k^S), i \geq 1, 1 \leq k \leq \mathbf{i}$	$(1 - p_{N_A}) \cdot \underline{S}^V(j_k^S - \sigma_1) \cdot \underline{\zeta}(u_3)$
$(i-1, j^A, \mathbf{k}^-, \ldots, j_{k-1}^S, j_{k+1}^S, \ldots, j^Q), \quad \mathcal{R}_2(j_k^S), 1 \leq i \leq M, 1 \leq k \leq \mathbf{i}$	$p_{N_A} \cdot \underline{S}^V(j_k^S - \sigma_1)$
$(i-1, j^A, \mathbf{k}, \ldots, j_{k-1}^S, \mathbf{v}_1, j_{k+1}^S, \ldots, j^Q), \quad \mathcal{R}_2(j_k^S), i > M, 1 \leq k \leq \mathbf{i}$	$p_{N_A} \cdot \underline{S}^V(j_k^S - \sigma_1) \cdot \underline{\beta}(u_1)$
Execution Stage 3:	
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_3, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_3(j_\mathbf{f}^S), \mathbf{v}_3 \neq j_\mathbf{f}^S, i \geq 1$	$\mathcal{S}^C(j_\mathbf{f}^S - \sigma_2, u_3)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_1, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_3(j_\mathbf{f}^S), i \geq 1$	$\widehat{p}_A(\mathbf{f}) \cdot \underline{S}^C(j_\mathbf{f}^S - \sigma_2) \cdot \underline{\beta}(u_1)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_4, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_3(j_\mathbf{f}^S), i \geq 1$	$(1 - \widehat{p}_A(\mathbf{f}))\widehat{p}_S(\mathbf{f}) \cdot \underline{S}^C(j_\mathbf{f}^S - \sigma_2) \cdot \underline{\beta}'_\mathbf{f}(u_4)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_5, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_3(j_\mathbf{f}^S), i \geq 1$	$(1 - \widehat{p}_A(\mathbf{f}))\widehat{p}_V(\mathbf{f}) \cdot \underline{S}^C(j_\mathbf{f}^S - \sigma_2) \cdot \underline{\eta}'_\mathbf{f}(u_5)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_6, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_3(j_\mathbf{f}^S), i \geq 1$	$(1 - \widehat{p}_A(\mathbf{f}))\widehat{p}_C(\mathbf{f}) \cdot \underline{S}^C(j_\mathbf{f}^S - \sigma_2) \cdot \underline{\zeta}'_\mathbf{f}(u_6)$
Execution Stage 4:	
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_4, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_4(j_\mathbf{f}^S), \mathbf{v}_4 \neq j_\mathbf{f}^S, i \geq 1$	$\mathcal{S}_\mathbf{f}^{S'}(j_\mathbf{f}^S - \sigma_3, u_4)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_5, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_4(j_\mathbf{f}^S), i \geq 1$	$\underline{S}_\mathbf{f}^{S'}(j_\mathbf{f}^S - \sigma_3) \cdot \underline{\eta}'_\mathbf{f}(u_5)$
Execution Stage 5:	
$(i, j^A, \mathbf{k}, \ldots, j_{k-1}^S, \mathbf{v}_5, j_{k+1}^S, \ldots, j^Q), \quad \mathcal{R}_5(j_k^S), \mathbf{v}_5 \neq j_k^S, i \geq 1, 1 \leq k \leq \mathbf{i}$	$\mathcal{S}_k^{V'}(j_k^S - \sigma_4, u_5)$
$(i, j^A, \mathbf{k}, \ldots, j_{k-1}^S, \mathbf{v}_6, j_{k+1}^S, \ldots, j^Q), \quad \mathcal{R}_5(j_k^S), i \geq 1, 1 \leq k \leq \mathbf{i}$	$\underline{S}_k^{V'}(j_k^S - \sigma_4) \cdot \underline{\zeta}'_k(u_6)$
Execution Stage 6:	
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_6, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_6(j_\mathbf{f}^S), \mathbf{v}_6 \neq j_\mathbf{f}^S, i \geq 1$	$\mathcal{S}_\mathbf{f}^{C'}(j_\mathbf{f}^S - \sigma_5, u_6)$
$(i, j^A, \mathbf{k}, \ldots, j_{\mathbf{f}-1}^S, \mathbf{v}_1, j_{\mathbf{f}+1}^S, \ldots, j^Q), \quad \mathcal{Q}(j^Q), \mathcal{R}_6(j_\mathbf{f}^S), i \geq 1$	$\underline{S}_\mathbf{f}^{C'}(j_\mathbf{f}^S - \sigma_5) \cdot \underline{\beta}(u_1)$

Table 1: Example of the Transition Rules for an Instance of our Generalized Parallel-Server Fork-Join Queueing System.

Table 1 specifically provides the rates of transitions from state $(i, j^A, \mathbf{k}, j_1^S, \ldots, j_{\mathbf{i}}^S, j^Q)$ to the states listed in the first column as a result of customer arrivals, timesharing quanta, task switches, and execution stages 1 through 6. The transition rates are provided in the second column in terms of the model parameters defined in Section 2 and above. Arrival transitions consist of three cases, the first representing internal transitions within the phase-type distribution PH($\underline{\alpha}, \mathcal{S}^A$) without an arrival, and the second and third representing arrivals according to $\underline{S}^A = -\mathcal{S}^A\mathbf{e}$ depending upon whether or not there are less than M customers at the single-server queue, respectively. Somewhat similarly, the fourth and sixth transitions in Table 1 represent internal timesharing quanta and task switch transitions within the phase-type distributions PH($\underline{\chi}, \mathcal{S}^Q$) and PH($\underline{\xi}, \mathcal{S}^O$), respectively. The fifth and seventh transitions represent completions of timesharing quanta and task switches according to $\underline{S}^Q = -\mathcal{S}^Q\mathbf{e}$ and $\underline{S}^O = -\mathcal{S}^O\mathbf{e}$, respectively, provided that there is at least one customer at the single-server queue. Execution stage 1 transitions consist of two cases, the first representing internal transitions within the phase-type distribution PH($\underline{\beta}, \mathcal{S}^S$) without a per-iteration service stage completion, and the second representing service stage completions according to $\underline{S}^S = -\mathcal{S}^S\mathbf{e}$ and initiation of a task vacation stage. (Recall that a task switch occurs on a vacation operation.) The first transitions of execution stage 2 represent internal transitions within the phase-type distribution PH($\underline{\eta}, \mathcal{S}^V$) without a per-iteration vacation stage completion, with the second transitions representing vacation stage completions according to $\underline{S}^V = -\mathcal{S}^V\mathbf{e}$ and initiation of a task communication stage, which occurs with probability $(1 - p_{N_A})$. With probability p_{N_A}, the customer departs the system at the end of the vacation stage as described in the third and fourth transitions of execution stage 2 that consist of vacation stage completions according to $\underline{S}^V = -\mathcal{S}^V\mathbf{e}$ depending upon whether there are M or more customers at the single-server queue, respectively. The first transitions of execution stage 3 represent internal transitions within the phase-type distribution PH($\underline{\zeta}, \mathcal{S}^C$) without a per-iteration communication stage completion, with the second transitions representing communication stage completions according to $\underline{S}^C = -\mathcal{S}^C\mathbf{e}$ and initiation of the service stage of the next iteration since the message to be received has already arrived, which occurs with probability $\widehat{p}_A(\mathbf{f})$. Otherwise, with probability $(1 - \widehat{p}_A(\mathbf{f}))$, the task waits for the message to arrive where the sender can be in the service, vacation or send stage, which occur with probability $\widehat{p}_S(\mathbf{f})$, $\widehat{p}_V(\mathbf{f})$ and $\widehat{p}_C(\mathbf{f})$, respectively (or, when $T_S > 0$, the sender can also be waiting for a message from yet another sibling parallel task; see Section 3.2.2). The transitions corresponding to these three cases represent the remaining transitions of execution stage 3 in Table 1. Similarly, the transitions corresponding to execution stages 4, 5 and 6 have analogous descriptions involving the phase-type distributions PH($\underline{\beta}'_k, \mathcal{S}_k^{S'}$), PH($\underline{\eta}'_k, \mathcal{S}_k^{V'}$) and PH($\underline{\zeta}'_k, \mathcal{S}_k^{C'}$). Additional details on the state transitions under each dynamic task scheduling policy are provided in the Appendix. Finally, note that the same logic (excluding execution stages 4, 5 and 6) is followed to construct the generator matrix $\mathbf{Q}^X$, although applied to the entire set of P single-server queues in all possible combinations.

3.2.2 Fixed-Point Iteration

Equations (8) through (14) form the solution to our decomposed single-server queue, in terms of the matrix $\mathbf{R}$, provided that certain aspects of the generalized parallel-server fork-join queueing system behavior are known. For the generic example of the previous section, these unknown measures include $\widehat{p}_A(\cdot)$, $\widehat{p}_S(\cdot)$, $\widehat{p}_V(\cdot)$, $\widehat{p}_C(\cdot)$, and the order ϕ_4, ϕ_5, ϕ_6 phase-type distributions PH($\underline{\beta}'_{(\cdot)}, \mathcal{S}_{(\cdot)}^{S'}$), PH($\underline{\eta}'_{(\cdot)}, \mathcal{S}_{(\cdot)}^{V'}$), PH($\underline{\zeta}'_{(\cdot)}, \mathcal{S}_{(\cdot)}^{C'}$) for execution stages 4, 5, 6. Estimates of these measures of the dynamic parallel-server system behavior are calculated in terms of the decomposed single-server queue solution, and a fixed-point iteration is used to obtain the final solution for the generalized parallel-server fork-join queueing system. We now describe this second aspect of our stochastic decomposition approach, initially focusing on the generic example of the previous section and then turning to the more general case.

Recall that, subject to timesharing quanta, task switch overheads and task scheduling dynamics in its service stages and communication stages, each parallel task alternates among service, vacation and communication stages in an iterative manner without delay provided that the required message to be received from a sibling parallel task as part of the communication stage has already arrived. On the other hand, when a given parallel task attempts to receive a message that has not yet arrived, then the task must wait for this message before proceeding, which may in turn (recursively) delay other sibling parallel tasks waiting for messages from the given parallel task.

Let $h(y)$ be the index of the state $y \in \Omega^Y$ in the lexicographic ordering of the elements of Ω^Y, and let

$$\mathbf{P}^Y \;=\; \mathbf{I} + \frac{\mathbf{Q}^Y}{\max\limits_k\{-\mathbf{Q}^Y[k,k]\}}$$

be the transition probability matrix for the uniformized version of the QBD process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$. We define

$$\mathcal{D}_\ell(k,u) \;\equiv\; \left\{\, y \;|\; y \in \Omega^Y \setminus \Omega_0^Y,\; \mathcal{R}_\ell(j_k^S),\; j_k^S = \sigma_{\ell-1} + u \,\right\}, \tag{17}$$

$$\widehat{\mathcal{D}}_\ell(k,u) \;\equiv\; \left\{\, z \;|\; z \in \Omega^Y,\; \mathbf{Q}^Y[h(z),h(y)] > 0,\; y \in \mathcal{D}_\ell(k,u) \,\right\}, \tag{18}$$

for $\ell = 1,\ldots,6$, $k = 1,\ldots,M$, $u = 1,\ldots,\phi_\ell$. Also, define

$$\mathcal{D}_\ell(k) \;\equiv\; \bigcup_{u=1}^{\phi_\ell} \mathcal{D}_\ell(k,u), \qquad\qquad \widehat{\mathcal{D}}_\ell(k) \;\equiv\; \bigcup_{u=1}^{\phi_\ell} \widehat{\mathcal{D}}_\ell(k,u), \tag{19}$$

for $\ell = 1,\ldots,6$, $k = 1,\ldots,M$, $u = 1,\ldots,\phi_\ell$.

To calculate the vectors $\widehat{p}_S$, $\widehat{p}_V$ and $\widehat{p}_C$ in terms of the decomposed stochastic process solution, recall that these measures represent the set of probabilities that the sender of the message being waited for (by the receiving parallel task being modeled) is in the service, vacation and send stage, respectively, at one of the other single-server queues. Due to the assumptions of stochastically independent and identical (in distribution) single-server queues as part of our approximate matrix-analytic solution, these probability vectors then can be expressed as

$$\widehat{p}_S(k) \;\propto\; (\boldsymbol{\pi}_0^Y,\ldots,\boldsymbol{\pi}_{M-1}^Y)\widehat{v}_{1,k}^b + \sum_{n=0}^{\infty} \boldsymbol{\pi}_{M+n}^Y \widehat{v}_{1,k}^r \;=\; (\boldsymbol{\pi}_0^Y,\ldots,\boldsymbol{\pi}_{M-1}^Y)\widehat{v}_{1,k}^b + \boldsymbol{\pi}_M^Y(\mathbf{I}-\mathbf{R})^{-1}\widehat{v}_{1,k}^r, \tag{20}$$

$$\widehat{p}_V(k) \;\propto\; (\boldsymbol{\pi}_0^Y,\ldots,\boldsymbol{\pi}_{M-1}^Y)\widehat{v}_{2,k}^b + \sum_{n=0}^{\infty} \boldsymbol{\pi}_{M+n}^Y \widehat{v}_{2,k}^r \;=\; (\boldsymbol{\pi}_0^Y,\ldots,\boldsymbol{\pi}_{M-1}^Y)\widehat{v}_{2,k}^b + \boldsymbol{\pi}_M^Y(\mathbf{I}-\mathbf{R})^{-1}\widehat{v}_{2,k}^r, \tag{21}$$

$$\widehat{p}_C(k) \;\propto\; (\boldsymbol{\pi}_0^Y,\ldots,\boldsymbol{\pi}_{M-1}^Y)\widehat{v}_{3,k}^b + \sum_{n=0}^{\infty} \boldsymbol{\pi}_{M+n}^Y \widehat{v}_{3,k}^r \;=\; (\boldsymbol{\pi}_0^Y,\ldots,\boldsymbol{\pi}_{M-1}^Y)\widehat{v}_{3,k}^b + \boldsymbol{\pi}_M^Y(\mathbf{I}-\mathbf{R})^{-1}\widehat{v}_{3,k}^r, \tag{22}$$

respectively, and appropriately normalized, where the n^{th} element of the vector $\widehat{v}_{\ell,k}^b$ (respectively, $\widehat{v}_{\ell,k}^r$) is 1 if the n^{th} state of the boundary (respectively, level M) is in $\mathcal{D}_\ell(k)$, and otherwise is 0, $\ell = 1,2,3$. The vector $\widehat{p}_A$ represents the probabilities that the modeled parallel task immediately enters the service stage of the next iteration upon completion of the communication stage because the message to be received has already arrived. Since $T_S = 0$, the sender of this message must either be in stages 4, 5 or 6, and thus we have the following expression for $\widehat{p}_A(k)$:

$$\widehat{p}_A(k) \;=\; \sum_{\ell=4}^{6} (\boldsymbol{\pi}_0^Y, \ldots, \boldsymbol{\pi}_{M-1}^Y)\, \widehat{v}_{\ell,k}^b \;+\; \boldsymbol{\pi}_M^Y (\mathbf{I} - \mathbf{R})^{-1}\, \widehat{v}_{\ell,k}^r. \tag{23}$$

Analogously, the initial probability vectors $\underline{\beta}'_k$, $\underline{\eta}'_k$ and $\underline{\zeta}'_k$ for the stage 4, 5 and 6 phase-type distributions can be written as

$$\underline{\beta}'_k(u) \;=\; \frac{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_1(k,u),\, y \in \mathcal{D}_1(k,u)} \pi^Y(z)\, \mathbf{P}^Y[h(z), h(y)]}{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_1(k),\, y \in \mathcal{D}_1(k)} \pi^Y(z)\, \mathbf{P}^Y[h(z), h(y)]}, \tag{24}$$

$$\underline{\eta}'_k(u) \;=\; \frac{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_2(k,u),\, y \in \mathcal{D}_2(k,u)} \pi^Y(z)\, \mathbf{P}^Y[h(z), h(y)]}{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_2(k),\, y \in \mathcal{D}_2(k)} \pi^Y(z)\, \mathbf{P}^Y[h(z), h(y)]}, \tag{25}$$

$$\underline{\zeta}'_k(u) \;=\; \frac{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_3(k,u),\, y \in \mathcal{D}_3(k,u)} \pi^Y(z)\, \mathbf{P}^Y[h(z), h(y)]}{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_3(k),\, y \in \mathcal{D}_3(k)} \pi^Y(z)\, \mathbf{P}^Y[h(z), h(y)]}, \tag{26}$$

respectively. These expressions can be easily represented in closed-form with the use of binary vectors in the same manner as above for the other probability measures.

Finally, the phase-type distributions for execution stages 4, 5 and 6 represent the times that the sender (of the message being waited for by the modeled receiving parallel task) spends in its service stage, in its vacation stage and in its communication stage, respectively. We construct these distributions from the decomposed single-server queue solution by examining the time until absorption in a set of new Markov processes $\{Y'(t)\,;\, t \in \mathbb{R}_+\}$ obtained from the original process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$ as follows. Let us first consider the phase-type distributions PH($\underline{\beta}'_k, \mathcal{S}_k^{S'}$) for stage 4. All of the states in $\mathcal{D}_2(k)$ are made to be absorbing states. We then make all of the remaining relevant states transient with the initial probability vector for every level i proportional to $\boldsymbol{\pi}_i^Y$ and appropriately normalized. The distributions PH($\underline{\beta}'_k, \mathcal{S}_k^{S'}$) then can be obtained either directly from the corresponding process $\{Y'(t)\,;\, t \in \mathbb{R}_+\}$ constructed in this fashion or by constructing a more compact phase-type distribution to match as many moments (and/or density function) of the time until absorption in this process $\{Y'(t)\,;\, t \in \mathbb{R}_+\}$ as are of interest, using any of the best known methods for doing so; e.g., see [1, 8, 15, 25, 24] and the references cited therein. The phase-type distributions PH($\underline{\eta}'_k, \mathcal{S}_k^{V'}$) and PH($\underline{\zeta}'_k, \mathcal{S}_k^{C'}$) for stages 5 and 6 are constructed in an exactly analogous manner. In each case, we capture the impact of timesharing quanta, task switches and other task scheduling dynamics on the execution stages of the sender task.

Remark 3.2. *Instead of constructing phase-type distributions for each of the stages 4, 5 and 6 individually, we can just as easily exploit our general approach above to construct phase-type distributions for the total* *time that the modeled parallel task spends waiting, given that the sender task is in its service stage, vacation stage or communication stage at the point when the parallel task attempts to receive the message. Then, with probability $\widehat{p}_S(\cdot)$, the parallel task waits according to phase-type distributions representing the sojourn times of the sender from its service stage through its communication stage. Similarly, with probability $\widehat{p}_V(\cdot)$, the parallel task waits according to phase-type distributions representing the sojourn times of the sender from its vacation stage through its communication stage. The phase-type distributions PH($\underline{\zeta}'_k, \mathcal{S}_k^{C'}$) would continue to be constructed as described above.*

Our stochastic decomposition approach to handling general instances of the generalized parallel-server fork-join queueing system depends in part upon the specific value of T_S for the queueing system of interest. When T_S is relatively small, we simply expand our approach above to capture the cases in which a series of up to T_S additional senders are waiting for messages from other single-server queues by repeating T_S times the execution stages 4, 5 and 6. For example, the ℓ^{th} set of waiting stages, consisting of execution stages $4 + 3\ell$, $5 + 3\ell$ and $6 + 3\ell$, are entered according to a set of rules analogous to those governing transitions from stage 3 to stages 4, 5 and 6 (e.g., refer to "Execution Stage 3" in Table 1) where each transition rule is multiplied by a probability proportional to $(1 - \widehat{p}_A(\mathbf{f}))^\ell$, which represents the probability that the sender of the message to be received by the parallel task being modeled is itself waiting on a series of ℓ servers to send their corresponding messages, $\ell = 0, \ldots, T_S$. A set of transition rules analogous to those governing transitions from stage 6 to stage 1 (e.g., see "Execution Stage 6" in Table 1) are also appropriately constructed for each of these sets of waiting stages to make the transition to the next set of waiting stages in the series. On the other hand, when T_S is relatively large, then we can simply use the execution stages 4, 5 and 6 of our approach above together with a form of the geometric distribution. More specifically, upon completing stage 6, the system returns to stages 4, 5 and 6 with probability $(1 - \widehat{p}_A(\mathbf{f}))$ (following appropriately modified versions of the set of rules governing transitions from stage 3 to stages 4, 5 and 6; e.g., refer to "Execution Stage 3" in Table 1) and otherwise enters stage 1 according to appropriately modified versions of the set of rules governing transitions from stage 6 to stage 1 (e.g., see "Execution Stage 6" in Table 1). Of course, both approaches can be used in combination. Lastly, the sending and receiving of multiple messages is accommodated as follows. The sending of multiple messages is simply incorporated in the corresponding phase-type distribution(s) of the single-server queue. Our stochastic decomposition approach is extended to handle the case of receiving messages from multiple tasks by replacing the expressions and arguments provided above with versions of these expressions and arguments based on the appropriate order statistics. As previously noted, in the case of all-to-all communication, we derive an expression for the distribution of time that the modeled parallel task waits for messages from all other tasks and use a phase-type distribution to estimate this waiting time distribution.

Now let $\kappa = (\widehat{p}_A(\cdot), \widehat{p}_S(\cdot), \widehat{p}_V(\cdot), \widehat{p}_C(\cdot), \underline{\beta}'_{(\cdot)}, \mathcal{S}_{(\cdot)}^{S'}, \underline{\eta}'_{(\cdot)}, \mathcal{S}_{(\cdot)}^{V'}, \underline{\zeta}'_{(\cdot)}, \mathcal{S}_{(\cdot)}^{C'})$. Notice that equations (20) – (26) and our general analysis above, which determine the value of κ, are expressed in terms of the decomposed single-server queue solution. Hence, we use a fixed-point iteration to solve for the invariant probability vector of the stochastic process. Initial values are chosen for κ and the components of the stationary probability vector $\boldsymbol{\pi}^Y$ are obtained using our matrix-analytic analysis. This solution yields new values for κ via the above equations and general analysis, and the single-server queue is solved again with these new values. This iterative procedure continues until the differences between the values of an iteration and those of the previous iteration are arbitrarily small. Numerous experiments have been performed with this fixed-point iteration. We note that the fixed-point iteration always converged quite rapidly, and that in all of the cases tested, changing the initial values had no effect on the calculated fixed-point, i.e., the single-server queue solution was insensitive to the initial values chosen for κ.

3.3 Performance Measures

Various performance measures of interest can be obtained from the components of the stationary probability vector $\boldsymbol{\pi}$. In particular, the expected number of customers in the generalized parallel-server fork-join queueing system can be expressed as

$$\mathbb{E}[\mathcal{N}] \;=\; \sum_{k=1}^{M-1} k\boldsymbol{\pi}_k\mathbf{e} + M\boldsymbol{\pi}_M \sum_{k=0}^{\infty}\mathbf{R}^k\mathbf{e} + \boldsymbol{\pi}_M \sum_{k=0}^{\infty} k\mathbf{R}^k\mathbf{e} \;=\; \sum_{k=1}^{M-1} k\boldsymbol{\pi}_k\mathbf{e} + M\boldsymbol{\pi}_M(\mathbf{I}-\mathbf{R})^{-1}\mathbf{e} + \boldsymbol{\pi}_M\mathbf{R}(\mathbf{I}-\mathbf{R})^{-2}\mathbf{e}. \tag{27}$$

The expected response time of a fork-join customer in the queueing system then can be calculated using Little's law [16] and (27), which yields

$$\mathbb{E}[\mathcal{T}] \;=\; \lambda^{-1}\left(\sum_{k=1}^{M-1} k\boldsymbol{\pi}_k\mathbf{e} \;+\; M\boldsymbol{\pi}_M(\mathbf{I}-\mathbf{R})^{-1}\mathbf{e} \;+\; \boldsymbol{\pi}_M\mathbf{R}(\mathbf{I}-\mathbf{R})^{-2}\mathbf{e}\right). \tag{28}$$

Another set of performance measures of interest is the long-run proportion of time that a single-server queue spends performing service, vacation, communication, task switching, and some form of waiting. These measures can be expressed as

$$\widetilde{p}^x \;=\; (\boldsymbol{\pi}_0,\,\ldots,\,\boldsymbol{\pi}_{M-1})\,v_b^x \;+\; \boldsymbol{\pi}_M(\mathbf{I}-\mathbf{R})^{-1}v_r^x, \tag{29}$$

where $x \in \{S, V, C, O, W\}$ such that the elements of this set are intended to indicate service (S), vacation (V), communication (C), task switching overhead (O) and waiting (W). Here we use the notation that the n^{th} position of the vector v_b^x (respectively, v_r^x) contains a 1 if the corresponding state of the boundary (respectively, level M) represents when the server is performing operations of type x, and otherwise contains a 0.

3.4 Resource Contention

The above analysis solves our generalized parallel-server fork-join queueing systems in the case where there is no resource contention. However, the performance of the entire queueing system and the execution of all parallel tasks may degrade when there is contention among the single-server queues for various resources. Given the dependency of such degradation in queueing performance on many detailed characteristics of the system in practice (including network bandwidth, server speed, and generalized fork-join customer execution patterns), we consider a very simple first-order approach to incorporate the effects of resource contention on queueing system performance in our analysis in a general manner that leaves system-specific details to be implicitly included in the resource contention model parameters. We shall focus on network contention to simplify the presentation, noting that the basic first-order approach can be extended to consider other sources of contention and their impact on queueing performance.

Let $f_d(k)$ denote the performance degradation factor for the generalized parallel-server fork-join queueing system when k of P servers are performing communication operations, $f_d(0) = 0$ and $f_d(k) < 1$, $k \in \{0, \ldots, P\}$. Namely, the percentage of the network capacity of the system, as a function of the number of single-server queues currently performing communication operations, is $100 \times (1 - f_d(k))\%$. In our exact analysis, we directly use $f_d(k)$ in all states where k single-server queues are performing communication operations by scaling the phase-type distribution PH$(\underline{\zeta}, \mathcal{S}^C)$ for the per-iteration communication durations of the fork-join customers in the analysis of Section 3.1 according to $\hat{\gamma} = \gamma(1 - f_d(k))$ such that $\hat{\gamma}^{-1} = -\underline{\zeta}(\mathcal{S}^C)^{-1}\mathbf{e}$. In our approximate analysis (cf. Section 3.2), due to our homogeneity assumptions, the long-run proportion of time that k of the P servers are performing communication operations is binomially distributed with parameters P and $\widetilde{p}^C$, rendering

$$q_P(k) \;=\; \left(\begin{array}{c} P \\ k \end{array}\right) (\widetilde{p}^C)^k (1 - \widetilde{p}^C)^{P-k}. \tag{30}$$

Using the functions $f_d(k)$ and $q_P(k)$, $k \in \{0, \ldots, P\}$, the average service rate of per-iteration communication stages can be expressed as

$$\hat{\gamma} \;=\; \frac{\gamma}{1 - q_P(0)} \sum_{k=1}^{P} q_P(k)(1 - f_d(k)), \tag{31}$$

where the normalization factor in (31) accounts for the fact that at least one single-server queue must be performing communication operations when the natural communication rate of the modeled single-server queue is γ. Hence, the potential system performance degradation due to network contention is incorporated in our approximate analysis by scaling the phase-type distribution PH$(\underline{\zeta}, \mathcal{S}^C)$ in the analysis of Section 3.2 according to equation (31) such that $\hat{\gamma}^{-1} \;=\; -\underline{\zeta}(\mathcal{S}^C)^{-1}\mathbf{e}$. The final solution of the resulting stochastic process then can be obtained by using appropriately modified versions of the fixed-point iteration.

4 Applications

Our general model of Section 2, under appropriate configuration, reduces to the classical fork-join queueing system, which has a rich history and has received considerable attention in the research literature; refer to, e.g., [21, 2, 30, 29, 6, 12] and the references therein. An exact mathematical analysis of the fork-join queue is often limited to relatively small P due to various factors, including the inherent complexity of the underlying stochastic process and the explosive growth of the state space with increasing P. Hence, a significant portion of previous research in the area is based on an approximate mathematical analysis; see, e.g., [21, 2, 30, 29, 6, 12].

In order to explore the accuracy of our approximate analysis of the generalized parallel-server fork-join queueing systems (cf. Section 3.2), a large number of numerical experiments have been conducted to compare the results of our approximate solution against detailed simulations across a wide range of model parameter assumptions and settings. The intensity of service, vacation and communication in a workload is designated by

$$\frac{\mu^{-1}}{\mu^{-1} + \nu^{-1} + \gamma^{-1}}, \qquad \frac{\nu^{-1}}{\mu^{-1} + \nu^{-1} + \gamma^{-1}}, \qquad \frac{\gamma^{-1}}{\mu^{-1} + \nu^{-1} + \gamma^{-1}},$$

respectively, where μ^{-1}, ν^{-1} and γ^{-1} are the mean service, vacation and communication durations of a parallel task in each iteration (cf. Section 2). This allows us to characterize a workload with respect to the degree of intensity for any of these execution stages.

We now briefly present a representative sample of the results from our numerical comparison experiments. Figure 1 presents relative errors of our approximate analysis as a function of the arrival rate λ for a generalized fork-join queueing system with $P = 32$ and all four dynamic task scheduling policies under service-intensive, communication-intensive and vacation-intensive workloads. In each case, our model is in good agreement with detailed simulations of the 32-server parallel system, often being within 5% and always less than 10%. Moreover, the accuracy of our approach increases with the size of the parallel-server queueing system, which is especially important for some of the large-scale systems motivating our study. Given the complex dynamic interactions among the different aspects of the generalized parallel-server queueing system and dynamic task scheduling policy, these results illustrate the potential benefits of our general stochastic decomposition approach for the approximate analysis of large-scale generalized parallel-server fork-join queueing systems.

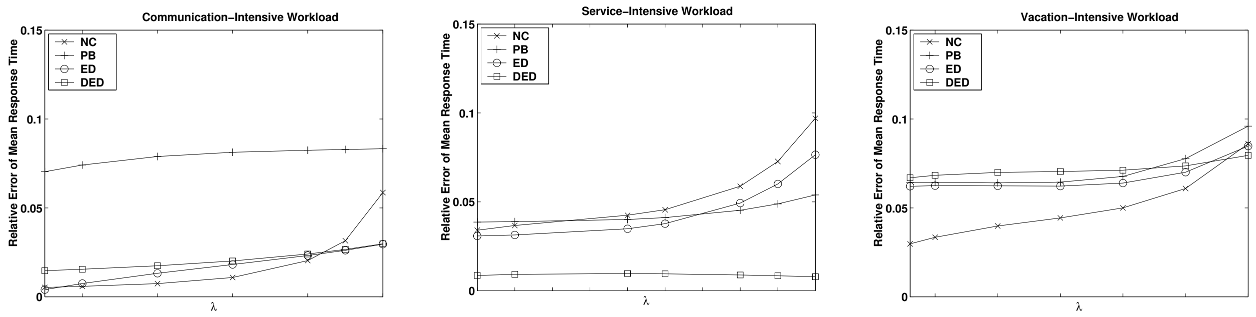

Figure 1: Results Comparing Approximate Solution Against Simulations

The accuracy of our approximate analysis of generalized parallel-server fork-join queueing systems with relatively small P is an important issue. Analogous to the well-known Erlang fixed-point approximation, which has received significant attention in the literature including issues related to accuracy outside of its limiting regime (refer to, e.g., [11] and the references therein), there can be model instances outside of the limiting regime where our approximation provides accurate results and other model instances outside of the limiting regime where our approximation provides inaccurate results. The primary interest in this paper has been on generalized fork-join queueing systems with large numbers of parallel servers, and this has influenced the derivation of our approximate analysis. Other approaches may be more appropriate for model instances with relatively small P when our approximate analysis is not sufficiently accurate.

Our numerical comparison experiments included the very simple first-order approach of Section 3.4 to approximate the impact of network contention on the performance measures of queueing systems with all four dynamic task scheduling policies. The main goal of the approach is to obtain a first-order approximation of the key performance trends that result from network contention. Despite the simplicity of this approach, our results demonstrate the same general trends obtained from measurements of comparable parallel computing systems with an overall degradation in performance caused by network contention; see, e.g., [17, 18, 31] and the references cited therein. This degradation in performance is most significant at moderate to fairly heavy traffic intensities, and especially for communication-intensive workloads. Furthermore, the general trends of our results are quite consistent with those of a previous study [28] where a similar approach was used together with computer measurements in the context of a stochastic model related to load balancing.

5 Conclusion

Motivated by a broad spectrum of evolving and new application domains, in this paper we have introduced a generalization of the classical parallel-server fork-join queueing system. The parallel tasks of each fork-join customer are significantly more complex, consisting of multiple iterations of a sequence of different types of execution stages which include task vacations and general communication among sibling tasks. The dynamics at each of the single-server queues are also significantly more complex including various scheduling policies to control the allocation of the server among the parallel tasks. Each customer departs the queueing system upon the completion of all of its multi-iteration tasks. We presented a formulation of the stochastic processes underlying these generalized parallel-server fork-join queueing systems, derived an exact matrix-analytic analysis that makes it possible to compute solutions for small queueing systems, and derived an approximate analysis by exploiting a stochastic decomposition, matrix-analytic analysis and fixed-point solution that makes it possible to compute solutions for large instances of the queueing system. Our approximate analysis is based on a general approach for solving complex multidimensional stochastic processes that consists of decomposing the dependencies and dynamic interactions among the different dimensions into simpler stochastic processes with reduced dimensionality and then approximating these dependencies and dynamic interactions through a system of fixed-point equations. The results of our exact and approximate analysis provide an important characterization of the probabilistic and dynamic behavior of the multidimensional stochastic processes underlying the generalized parallel-server fork-join queueing systems with dynamic task scheduling.

Given the dynamic and complex interactions among the single-server queues, the generalized fork-join customers and the dynamic task scheduling policy, it can be both difficult and expensive to either experiment with or simulate such parallel-server queueing systems as they arise in various application domains. We are therefore particularly interested in a limiting regime based on generalized fork-join queueing systems with large numbers of parallel servers, especially given a broad spectrum of new and emerging applications comprised of thousands and even tens of thousands of computing servers. Numerical comparison experiments exhibit good agreement between the solutions of our approximate analysis and detailed simulations, with the results demonstrating that the accuracy of our approach increases as the number of parallel servers grows.

Acknowledgements

The authors thank Peter Taylor and the referees for helpful comments on an earlier version of this paper.

References

[1] S. Asmussen, O. Nerman, and M. Olsson. Fitting phase-type distributions via the EM algorithm. *Scandinavian Journal of Statistics*, 23:419–441, 1996.

[2] F. Baccelli, A. M. Makowski, and A. Shwartz. The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds. *Advances in Applied Probability*, 21:629–660, 1989.

[3] L. Bright and P. Taylor. Calculating the equilibrium distribution in level dependent quasi-birth-and-death processes. *Stochastic Models*, 11:497–526, 1995.

[4] L. Bright and P. Taylor. Equilibrium distributions for level-dependent quasi-birth-and-death processes. In S. R. Chakravarthy and A. S. Alfa, editors, *Matrix-Analytic Methods in Stochastic Models*, pages 359–375. Marcel Dekker, 1997. Lecture Notes in Pure and Applied Mathematics, Vol. 183.

[5] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 67–90. Springer-Verlag, 1999. Lecture Notes in Computer Science Vol. 1659.

[6] R. J. Chen. A hybrid solution of fork/join synchronization in parallel queues. *IEEE Transactions on Parallel and Distributed Systems*, 12:829–845, 2001.

[7] A. C. Dusseau, R. H. Arpaci, and D. E. Culler. Effective Distributed Scheduling of Parallel Workloads. In *Proceedings of the ACM SIGMETRICS 1996 Conference on Measurement and Modeling of Computer Systems*, pages 25–36, 1996.

[8] A. Feldmann and W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation*, 31:245–279, 1998.

[9] D. Gamarnik, N. Jengte, Y. Lu, B. Ramachandran, M. S. Squillante, A. Radovanovic, J. Benayon, and V. Szaloky. Analysis of business processes using queueing analytics. Preprint, 2006.

[10] N. Islam, A. Prodromidis, and M. S. Squillante. Dynamic partitioning in different distributed-memory environments. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 244–270. Springer-Verlag, 1996. Lecture Notes in Computer Science Vol. 1162.

[11] F. P. Kelly. Loss networks. *Annals of Applied Probability*, 1(3):319–378, 1991.

[12] S.-S. Ko and R. F. Serfozo. Response times in M/M/s fork-join networks. *Advances in Applied Probability*, 36(3):854–871, 2004.

[13] G. Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi-birth-and-death processes. *Journal of Applied Probability*, 30:650–674, 1993.

[14] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[15] G. Latouche and P. Taylor. *Advances in Algorithmic Methods for Stochastic Models*. Notable, 2000.

[16] J. D. C. Little. A proof of the queuing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[17] S. Nagar, A. Banerjee, A. Sivasubramaniam, and C. R. Das. A Closer Look at Coscheduling Approaches for a Network of Workstations. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 96–105, June 1999.

[18] S. Nagar, A. Banerjee, A. Sivasubramaniam, and C. R. Das. Alternatives to Coscheduling a Network of Workstations. *Journal of Parallel and Distributed Computing*, 59(2):302–327, November 1999.

[19] V. K. Naik, S. K. Setia, and M. S. Squillante. Processor allocation in multiprogrammed, distributed-memory parallel computer systems. *Journal of Parallel and Distributed Computing*, 46(1):28–47, October 1997.

[20] R. D. Nelson and M. S. Squillante. Parallel-server stochastic systems with dynamic affinity scheduling and load balancing. Preprint, 2006.

[21] R. D. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, June 1988.

[22] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. The Johns Hopkins University Press, 1981.

[23] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, 1989.

[24] A. Riska, V. Diev, and E. Smirni. An EM-based technique for approximating long-tailed data sets with PH distributions. *Performance Evaluation*, 55(1-2):147–164, 2004.

[25] A. Riska, M. S. Squillante, S.-Z. Yu, Z. Liu, and L. Zhang. Matrix-analytic analysis of a MAP/PH/1 queue fitted to Web server data. In G. Latouche and P. Taylor, editors, *Matrix-Analytic Methods: Theory and Applications*, pages 333–356. World Scientific, 2002.

[26] P. G. Sobalvarro. *Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, January 1997.

[27] M. S. Squillante. Stochastic analysis of multiserver systems. *Performance Evaluation Review*, 34(4), 2007. ACM Press.

[28] M. S. Squillante and R. D. Nelson. Analysis of task migration in shared-memory multiprocessors. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 143–155. ACM, May 1991.

[29] X. Tan and C. Knessl. A fork-join queueing model: Diffusion approximation, integral representations and asymptotics. *Queueing Systems Theory and Applications*, 22:287–322, 1996.

[30] S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. *Performance Evaluation*, 20:245–265, 1994.

[31] Y. Zhang, A. Sivasubramaniam, J. Moreira, and H. Franke. A Simulation-based Study of Scheduling Mechanisms for a Dynamic Cluster Environment. In *Proceedings of the ACM 2000 International Conference on Supercomputing*, pages 100–109, May 2000.

A Appendix

A key aspect of our exact and approximate solutions of the proposed generalized parallel-server fork-join queueing systems is the formulation of the corresponding generator matrices $\mathbf{Q}^X$ and $\mathbf{Q}^Y$, respectively. This is especially the case in our approximate analysis that is based on a general form of stochastic decomposition. To further elucidate the exposition of our analysis, in this appendix we present an overview of these additional details for our decomposed stochastic process of each single-server queue. In the interest of clarity, we shall assume an exponential distribution for each stage of execution and use a random communication pattern between sibling parallel tasks, i.e., during each iteration, a parallel task only sends and receives one message, and its receiver and sender sibling parallel tasks are randomly chosen. We also assume that, at any time, the two communicating sibling parallel tasks are not more than one iteration apart, with instantaneous message transmission times. The corresponding details for our general model of Section 2 and analysis of Section 3 can be obtained in a straightforward manner from the details that follow.

Under our approximate analysis, a parallel task goes through six execution stages from the viewpoint of the fork-join customer behavior: service (with probability $\widehat{p}_S(\cdot)$), vacation (with probability $\widehat{p}_V(\cdot)$), sending messages (with probability $\widehat{p}_C(\cdot)$), waiting for messages from sibling parallel tasks in service, waiting for messages from sibling parallel tasks on vacation, and waiting for messages from sibling parallel tasks sending them. From the viewpoint of the single-server queue, a parallel task generally can be in one of three states: executing, ready (i.e., available to execute at the server but waiting in the (local) queue), or blocked. The combination of these two perspectives determine the state of a parallel task at any instant of time (as shown in Figure 2). (Note that a parallel task must be blocked when on vacation.) A parallel task generally moves to a ready state from executing whenever it is switched out (shown as transitions from state S_i to state S'_i in Figure 2) at the rate τ of timesharing quantum expiration. The reverse transition to the executing state (from ready) depends on the number and states of other parallel tasks at the single-server queue (shown as transitions from state S'_i to state S_i). (Note that ED and DED will transition a parallel task from state S'_i to S_i for other reasons as well.) The transition rates from state S_1 to S_2 and from state S_2 to S_3 are simply a function of the service and vacation rates (μ and ν), respectively.

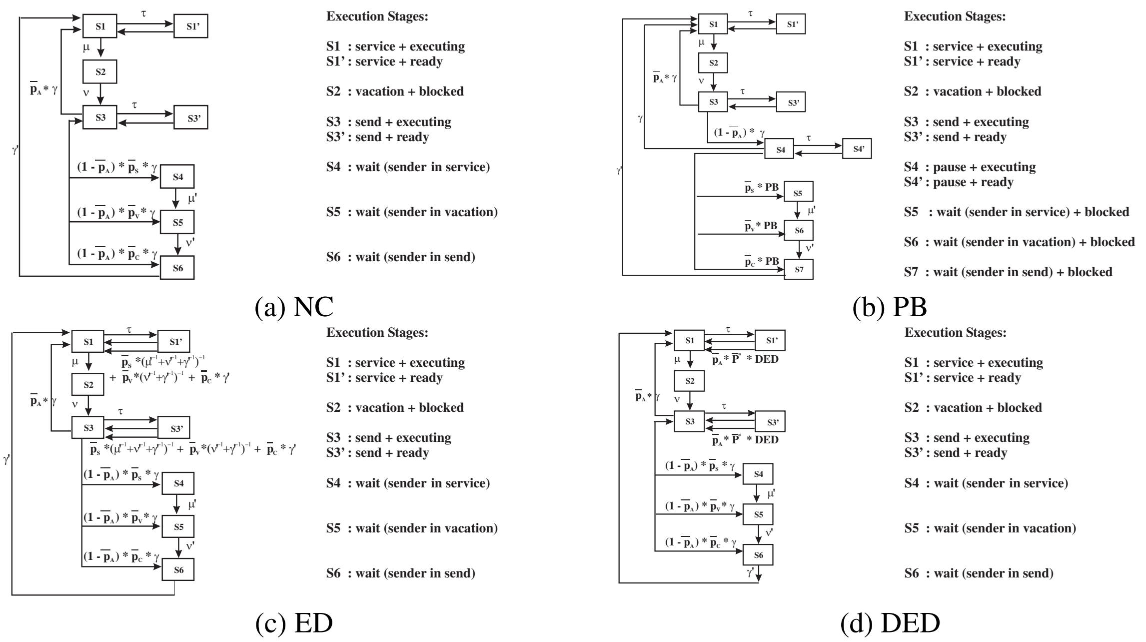

Figure 2: State transitions for the four dynamic task scheduling policies.

The above transitions are common to all four dynamic task scheduling policies of Section 2. We next turn to consider the remaining transitions that are specific to each of these task scheduling policies.

No Coordination (NC). Figure 2(a) illustrates the state transitions for NC. Once a parallel task leaves state S_3 (finishes sending messages), it either moves to state S_1 (starts next iteration) if the message it will receive is available as determined by the probability $\widehat{p}_A$, or it starts waiting for the message in one of the states S_4, S_5 or S_6 depending on the state of the sender sibling parallel task with probabilities $(1-\widehat{p}_A) \times \widehat{p}_S$, $(1-\widehat{p}_A) \times \widehat{p}_V$, or $(1-\widehat{p}_A) \times \widehat{p}_C$, respectively. The value of $\widehat{p}_A$ can be calculated using equation (23). The transitions from state S_4 to S_5, and from state S_5 to S_6 are the same as transitions from S_1 to S_2 (though with rate μ') and from S_2 to S_3 (though with rate ν'), respectively. Once the sender sibling parallel task is in the sending state, the parallel task being modeled will leave state S_6 at rate γ', and move to state S_1 (start next iteration). Note that other aspects of the modeled parallel task may change (e.g., whether or not the task possesses the server) while it is in S_4, S_5 or S_6 (not shown).

Pause and Block (PB). Figure 2(b) illustrates the state transitions for PB. A parallel task goes through the sequence of service (S_1 and S_1'), vacation (S_2), sending messages (S_3 and S_3'), and waiting for messages. The stage of waiting for messages is further broken down into five states: pausing (S_4 and S_4'), waiting (while blocked) for messages from a sibling parallel task that is in service (state S_5), from a sender that is on vacation (state S_6), and from a sender that is sending messages (state S_7). In addition to the transitions for NC, two more transitions are noteworthy for PB. In state S_4, the parallel task pauses for mean time PB^{-1} and if the message has not arrived within that time, then it will block itself and transition to one of the three states: S_5, S_6 and S_7 depending on the state of the sending sibling parallel task. A subsequent message arrival will transition the parallel task out of the blocked state as shown by the transition from state S_7 to S_1.

Event Driven (ED). The main differences between ED, whose state transitions are shown in Figure 2(c), and NC (compare with Figure 2(a)) are in that a message arrival when the parallel task is waiting for a message immediately takes it to the next iteration. For this reason, the priority increase can again affect the rates at which the parallel task transitions from some of the ready states back to the corresponding executing states, i.e., from states S_1' and S_3' to S_1 and S_3, respectively. Assuming an exponential distribution for each stage, we can calculate the rate of message arrival before the modeled parallel task starts to wait for the message as

$$\widehat{p}_S \, \frac{1}{\mu'^{-1} \, + \, \nu'^{-1} \, + \, \gamma'^{-1}} \, + \, \widehat{p}_V \, \frac{1}{\nu'^{-1} \, + \, \gamma'^{-1}} \, + \, \widehat{p}_C \, \gamma'.$$

Note that other aspects of the modeled parallel task may change (e.g., whether or not the task possesses the server) while it is in S_4, S_5 or S_6 (not shown).

Delayed Event Driven (DED). Figure 2(d) illustrates the state transitions for a parallel task under DED. The states remain the same as in NC, with the only differences in transition rates being due to the periodic activity which can increase priorities. These differences will affect the transitions from ready to executing states when the parallel task is in service and sending stages (from states S_1' and S_3' to states S_1 and S_3, respectively) as was the case for ED. This rate is defined as $\widehat{p}_A \times \widehat{P}^* \times \rho$, where $\widehat{p}_A$ is the probability of this parallel task having a pending message (calculated by equation (23)) when the periodic activity takes place, $\widehat{P}^*$ is the probability that all the other parallel tasks at the single-server queue which have higher priority than this parallel task do not have a pending message (i.e., $(1-\widehat{p}_A)^n$ if there are n such tasks), and DED denoting the frequency for this periodic activity. Note that other aspects of the modeled parallel task may change (e.g., whether or not the task possesses the server) while it is in S_4, S_5 or S_6 (not shown).