Chalmers Publication Library

Shared-object System Equilibria: Delay and Throughput Analysis

(article starts on next page)

Shared-object System Equilibria:
Delay and Throughput Analysis
(Technical Report)

Iosif Salem   Elad M. Schiller   Marina Papatriantafilou   Philippas Tsigas

Department of Computer Science and Engineering
Chalmers University of Technology
S–412 96, Göteborg, Sweden
{iosif, elad, ptrianta, tsigas}@chalmers.se

Abstract

We consider shared-object systems that require their threads to fulfill the system jobs by first acquiring sequentially the objects needed for the jobs and then holding on to them until the job completion. Such systems are in the core of a variety of shared-resource allocation and synchronization systems. This work opens a new perspective to study the expected job delay and throughput analytically, given the possible set of jobs that may join the system dynamically. We identify the system dependencies that cause contention among the threads as they try to acquire the job objects. We use these observations to define the shared-object system equilibria. We note that the system is in equilibrium whenever the rate in which jobs arrive at the system matches the job completion rate. These equilibria consider not only the job delay but also the job throughput, as well as the time in which each thread blocks other threads in order to complete its job. We then further study in detail the thread work cycles and, by using a graph representation of the problem, we are able to propose procedures for finding and estimating equilibria, i.e., discovering the job delay and throughput, as well as the blocking time. To the best of our knowledge, this is a new perspective, that can provide better analytical tools for the problem, in order to estimate performance measures similar to ones that can be acquired through experimentation on working systems and simulations, e.g., as job delay and throughput in (distributed) shared-object systems.

1 Introduction

We consider shared-object systems that require their threads to fulfill the system jobs by first acquiring sequentially all of the job objects. The job then holds on to these objects until the job operation is done. We identify the system dependencies that cause contention among the threads as they try to acquire the job objects. We study the (stochastic) processes of job arrival and completion with an emphasis on the cases in which the job arrival rate matches the job completion rate, i.e., the job throughput. In these cases, the system is in a *shared-Object System Equilibrium* (OSE). For a given $\varepsilon > 0$ and an OSE, we say that the system is in an ε-*OSE* when the completion rate of any job differs from the one of an OSE by at most ε. We study the conditions for a given

1

shared-object system to be in an OSE as well as contention-related properties of OSEs, i.e., the *expected* job delay and completion rate, as well as the time in which each thread blocks other threads and by that prevents them from making progress. We propose an analytical procedure for finding (in polynomial time) ε-OSEs. Moreover, we estimate the performance measures of systems that are in ε-OSE.

The existing practice considers job delay and completion rate as the performance measures of working systems. Empirical experiments often study shared-resource systems at their saturation point in which the system is at its peak utilization. Let us describe peak utilization scenarios using two vectors; one for job arrival rates and another for their completion rates. A saturation point is the case in which: (1) the system is in equilibrium, i.e., the arrival rate of any particular job matches the completion rate of this job, as well as (2) the system is at the stage at which a higher arrival rate of any job to the system cannot increase its completion rate. Our study considers the entire range of these equilibria rather than just peak utilization scenarios (Section 2). We then propose a procedure for finding ε-OSEs, if such exist in the given system (Section 3). Once we find an ε-OSE, we can estimate its performance measures, i.e., job delay, completion rate and blocking time. To this end, we develop a number of analytical tools for OSEs. Given the job arrival rates, we show how to estimate the probabilities for threads to follow a certain object acquisition sequence (Section 5). We are then able to formulate recursive equations (with interdependencies) for calculating the blocking periods and the completion rates (sections 6, and respectively, 7). We overcome these dependencies and solve these recursive equations by analysing the thread work cycles (Section 8).

Related Work. Our problem domain considers computing entities, which are called threads. Each thread runs a sequential program that has to acquire reusable resources (objects), often several at the same time, for a bounded time of use. To guarantee deadlock absence, it is important that all threads acquire the objects in an ordered manner. For example, one can deterministically define a partial order among the objects, such that the threads acquire them in totally ordered manner. We consider a generalization of the dining philosophers problem, as in [15, 17], in which every job includes a fixed set of objects that it may need. This problem has well-known results studying the worst-case job delays, which may even be exponential on metrics, such as the chromatic number of the resource graph [14, 15]. In this graph, the vertices (objects) are connected if there is at least one thread that may request them both at any point in time. In the context of actual systems, the expected time is rather different than the worst case and therefore computer experiments are the common way for evaluating the system performance. We provide a new perspective that enables an analysis of the evaluation metrics by considering measures both at the system level and at the level of each resource. In particular, we consider performance measures that are associated with each resource, such as the delay, completion rate and blocking time. On the system level, we consider the job arrival and completion rates, as well as the total number of threads, N, and objects, M.

Our contribution. We study analytical tools that provide the means to estimate performance measures of working distributed systems. In the context of synchronization challenges that are modeled via a generalization of the dynamic dining philosophers problem, our analytical tools are the first, to the best of our knowledge, to consider performance measures similar to the ones that can be acquired via experimentation on working systems and simulations.

For a given number of threads and job arrival rates, we provide a way to analyze the delay of jobs and their completion rates as well as the time for which the threads are blocked. In addition to the job completion period (Lemma 3), we analyze a number of key properties, such as the probability to request a particular resource after the acquisition of another specific resource, the time during which threads that have acquired such a particular resource block other threads that ask to access the same

2

resource (Lemma 4) as well as the time between two requests to access such resources (Lemma 5). Since these properties have interdependencies due to thread blocking, we show how the concept of thread work cycles can be represented in subsystems that also include such interdependencies but have no thread blocking. This way, we can resolve these interdependencies (Theorem 6) and estimate the performance of the given (distributed) shared resource system. Moreover, we use the work cycle events (sections 2.3 and 2.5) to verify our modeling approach. We present a procedure for satisfying approximately the equilibrium conditions and by that find an ε-OSE as well as the performance measures of the studied system (Section 3).

Our contribution can facilitate early-stage evaluations of systems that are similar to the studied one. Moreover, using our proposed methods, one can analytically, rather than via empirical experiments, study trade-offs among OSEs. Such trade-offs can facilitate the design of mechanisms for adjusting the number of threads and job arrival rates according to the performance measures of a dynamic system.

2   Preliminaries

We consider a system that includes *(system) items*, which are *(totally ordered) objects*, ($object[1]$, ..., $object[M]$), and *(totally ordered) threads*, ($thread[1]$, ..., $thread[N]$). The objects are shared in a mutually exclusive way, i.e., only one thread at a time may gain access to an object. Each thread is to carry out one *job* at a time, where $job_i = \langle objs_i, operation_i \rangle$, J is the number of the system's jobs, $i \in [1, J]$ and $objs_i = (object_{i_1}, ..., object_{i_k})$ is an arbitrary, non-empty subsequence of ($object[1], ..., object[M]$), and thus $objs_i$ follows the same order. Note that we assume that $objs_i$ is a fixed vector and that different jobs may have different object vectors of different lengths. Moreover, the *(job) operation time*, O_i, is a random variable with a known distribution. Namely, we assume that the time it takes to execute the job operation is provided, say, via a profiler.

2.1   Acquisition paths, periods and requests

Suppose that the system assigns job_i to $thread[n] : 1 \leq n \leq N$. In this case, job_i's *(acquisition) path* is the vector, $path_{i,n} = (thread[n], object[i_1], ..., object[i_k])$, in which $thread[n]$ carries out job_i's operation after it has sequentially acquired $object_{i_1}, ..., object_{i_k}$. A thread can acquire a particular object, $object[i]$, by pending its *(acquisition) request* in a (first in, first out) queue $Q(object[i])$ until all (previously) waiting threads in $Q(object[i])$ have acquired and released $object[i]$. The *acquisition period*, A, is a known random variable that refers to a period that starts when a thread has acquired an object (or just been assigned to a new job) and ends as soon as that thread places a request for the next object. Namely, we assume that the time it takes to send a request after a supply event is provided, say, via a profiler. Once the thread sequentially acquires the entire object set, $object_{i_1}$, ..., $object_{i_k}$, it executes the job operation, $operation_i$, before completing the job. We say that a thread is *blocking* when other threads are queuing for its acquired objects. That happens whenever different jobs have overlapping object vectors. Note however that threads carry out jobs within finite time even in the presence of blocking, because our definition of acquisition paths considers object acquisition according to a common (total) order. This work focuses on systems that can be in an equilibrium and while in equilibrium it holds that the number of pending requests in $Q(object[i]) : i \in [1, M]$ is bounded.

2.2 Job arrival rates

We assume that the time between two consecutive arrivals of job_i to $thread[n]$ is a random variable $I[i, n]$ (inter-arrival period), where $i \in [1, J]$ and $n \in [1, N]$. We define the *job arrival rate*, $\lambda_{i,n}$, in which job_i arrives at the system that then places job_i in a (first in, first out) queue, $Q(thread[n])$, where $\lambda_{i,n}$ is a positive real number. The inter-arrival period of $I[i, n]$ follows an exponential distribution, $Exp(\lambda_{i,n})$. Note that this is a common way to model arrivals, e.g. [7]. As soon as $thread[n]$ becomes available, the system assigns to $thread[n]$ the job that is in $Q(thread[n])$'s top. This work focuses on systems that can be in an equilibrium for which the number of pending jobs in $Q(thread[n])$ is bounded.

2.3 Work cycles: demand, supply and release

The thread work cycle, $cycle(thread[n], job_i)$, refers to the events that occur during the period that starts when the system assigns job_i to $thread[n]$ and ends immediately before the next assignment of any job to $thread[n]$. It starts with the event $\sigma_i(thread[n])$ in which the system assigns job_i to $thread[n]$. It also includes the events in which: $thread[n]$ *demands* (requests) access to $object[k]$, denoted by $\delta_i(thread[n], object[k])$, the system *supplies* (provides) access to $object[k]$, denoted by $\sigma_i(object[k], thread[n])$, and the event in which $thread[n]$ releases $object[k]$ denoted by $\phi_i(object[k], thread[n])$. For simplicity, we refer to the sequence of these release events $\Phi_i(thread[n]) = (\phi_i(object[k_1], thread[n]), \ldots, \phi_i(object[k_\ell], thread[n]))$ as a single event and assume that $thread[n]$ releases all its acquired objects instantaneously and immediately after the operation time, O_i, which is a random variable. Immediately after the event $\Phi_i(thread[n])$, the thread work cycle starts a (possibly zero length) *idle period*, before the system assigns the next (and possibly different than the previous) job to $thread[n]$ so that the next work cycle begins.

We assume that events are instantaneous and mark them as points on a thread's work cycle (Figure 1). Note however, that between a supply event and a demand event (as well as the last supply event and the release event), there is a random length period, i.e. the (random) acquisition period A (and the operation time O_i, respectively), which refers to scheduling uncertainties. Hence, we denote $thread[n]$'s work cycle due to job_i as $cycle(thread[n], job_i) \equiv (\sigma_i(thread[n]), \delta_i(thread[n], object[k_1]), \sigma_i(object[k_1], thread[n]), \ldots, \delta_i(thread[n], object[k_\ell]), \sigma_i(object[k_\ell], thread[n]), \Phi_i(thread[n]))$.

Figure 1: The thread work cycle

2.4 Subpaths and acquisition graph

Let s (source) and d (destination) be (possibly consecutive) items on a path. We define the set $\epsilon(s, d) = \{(s, \bullet, d) \mid \exists job_i : path_i = (\bullet, s, \bullet, d, \bullet)\}$ of all *subpaths* between s and d, where $\bullet$ denotes a finite, possibly empty, item sequence. The *acquisition graph*, $\mathscr{G} = (V, E)$, is a simple directed graph, where $V = \{thread[1], \ldots, thread[N], object[1], \ldots, object[M]\}$ are the system items (Figure 2). The edges $E = \{(s, d) \mid \exists job_i : path_i = (\bullet, s, d, \bullet)\}$ are two consecutive items on a path.

2.5 Conditional and consecutive events

We consider an event that occurs at item d (destination) in condition to an event occurrence at item s (source), where $(\bullet, s, d, \bullet)$ is a subpath of job_i and both events belong to the same work cycle of job_i that $thread[n]$ carries out. **Conditional demand and supply events.** Denote by $\delta_{i,n}(d \mid s) \equiv \langle \delta_i(thread[n], d) \mid \sigma_i(s, thread[n]) \rangle$ the *conditional (demand) event*, $\delta_i(thread[n], d)$, in which $thread[n]$ requests access to object d immediately after the supply event, $\sigma_i(s, thread[n])$. Note that the event $\sigma_i(s, thread[n])$ may refer to: (1) access to object s, or (2) job_i's assignment to $s = thread[n]$ (Figure 1), i.e., $\sigma_i(thread[n]) \equiv \sigma_i(thread[n], thread[n])$. E.g., $\delta_{i,n}(object[j] \mid object[k])$ denotes the conditional demand event $\langle \delta_i(thread[n], object[j]) \mid \sigma_i(object[k], thread[n]) \rangle$ in which $thread[n]$ requests access to $object[j]$ im-

Figure 2: An acqui-sition graph, $\mathscr{G}$

mediately after gaining access to $object[k]$, where $k \in [1, M-1]$, $j \in (k, M]$ and $n \in [1, N]$. Another example is the case $\delta_{i,n}(object[j] \mid thread[n])$, where the conditional demand event $\langle \delta_i(thread[n], object[j]) \mid \sigma_i(thread[n], thread[n]) \rangle$ refers to $thread[n]$'s request to access job_i's first object, $object[j]$, immediately after the assignment of job_i to $thread[n]$. In a similar manner, denote by $\phi_{i,n}(d|s) \equiv \langle \phi_i(d, thread[n]) \mid \sigma_i(s, thread[n]) \rangle$ the *conditional (release) event*, in which $thread[n]$ releases object d at event $\phi_i(d, thread[n])$ that occurs after the supply event, $\sigma_i(s, thread[n])$ and at the same work cycle (we will mainly use $\phi_{i,n}(s|s)$).
Events of arbitrary jobs and threads. Sometimes we consider an arbitrary job_i that an arbitrary $thread[n]$ carries out. We then write $\delta(d|s)$, $\sigma(s)$ and $\phi(d|s)$ instead of $\delta_{\bullet,\bullet}(d|s)$, $\sigma_{\bullet,\bullet}(s)$, and respectively, $\phi_{\bullet,\bullet}(d|s)$ when referring to events from the sets $\{\delta_{i,n}(d|s) : i \in [1, J], n \in [1, N]\}$, $\{\sigma_{i,n}(s) : i \in [1, J], n \in [1, N]\}$, and respectively, $\{\phi_{i,n}(d|s) : i \in [1, J], n \in [1, N]\}$, where $i \in [1, J]$ and $n \in [1, N]$. Given subpath $(\bullet, \ell_1, \ldots, \ell_k, \bullet)$, denote by $\delta(\ell_k | \ell_1, \ldots, \ell_{k-1})$ the occurrence of $\delta(\ell_k | \ell_{k-1})$, which happens immediately after $\delta(\ell_{k-1} | \ell_{k-2})$, $\ldots$, $\delta(\ell_2 | \ell_1)$, where $k \in [1, M]$.
Consecutive events. Let $i, j \in [1, J]$, $k, \ell \in [1, M]$, $n, n' \in [1, N]$, $\delta_{i,n}(object[k]|s)$ and $\delta_{j,n'}(object[\ell]|s)$. We say that the event $\delta_{i,n}(object[k] \mid s)$ *occurs consecutively after* the event $\delta_{j,n'}(object[\ell]|s)$, when $\delta_{j,n'}(object[\ell] \mid s) = \langle \delta_j(thread[n'], object[\ell]) \mid \sigma_j(s, thread[n']) \rangle$ is the first conditional demand event, that includes the supply event $\sigma_j(s, thread[n'])$, to occur after the conditional demand event $\delta_{i,n}(object[k] \mid s) = \langle \delta_i(thread[n], object[k]) \mid \sigma_i(s, thread[n]) \rangle$, that includes the supply event $\sigma_i(s, thread[n])$.

2.6 Pairwise states and request probabilities

The definition of the studied equilibria (at the system level) is based on item-level definitions that consider $\mathscr{G}$'s edges, $(s, d) \in E$. We present a definition of the (pairwise) state, $c[s, d]$, which considers the delay, blocking and inter-demand periods that are related to the edge (s, d) and its conditional events. These periods refer to the time it takes threads to request access to object d, and release it subsequently (after the acquisition of item s) as well as the time between such requests that are made by (possibly) different threads. Moreover, when estimating the value of the pairwise-state, $c[s, d]$, we need to consider the probabilities that are related to the edge (s, d) and its conditional events.
Pairwise states. We refer to (s, d)'s *pairwise state* $c[s, d] = s.\langle T[k], D[k], B[k] \rangle$ as the tuple that includes the request completion rate, request delay, and respectively, blocking period with relation to the events $\delta(d|s)$, where $d = object[k]$ is an object and $k \in [1, M]$. The (s, d)'s *request inter-demand period, $s.T[k]$,* refers to the period between the consecutive events, $\langle \delta_i(thread[n], d)$
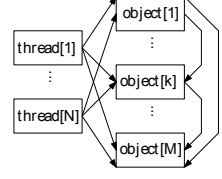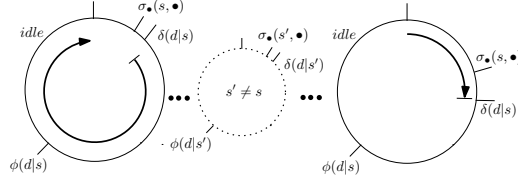
Figure 3: (s, d)'s inter-demand period, $s.T[k]$, $d = object[k]$, i.e., the period between two consecutive $\delta(d|s)$ events.

$| \; \sigma_i(s, \; thread[n])\rangle$ and $\langle\delta_j(thread[n'], \; d) \; | \; \sigma_j(s, \; thread[n'])\rangle$ (Figure 3), where $i, j \in [1, J]$ and $n, n' \in [1, N]$. Notice that we can estimate the throughput, $1/s.T[k]$, which is the number of $\delta(d|s)$ requests per time unit. Moreover, we can consider the expected throughput, $1/E(s.T[k])$, where $E(s.T[k])$ is the expected value of $s.T[k]$. Furthermore, (s, d)'s delay, $s.D[k]$, refers to a period that starts on the event $\delta(d|s)$ in which a thread requests access to object d (immediately after gaining access to item s) and ends upon the event $\phi(d|s)$ in which that thread releases object d (during the same work cycle). In addition, (s, d)'s blocking, $s.B[k]$, is the fraction of (s, d)'s delay $s.D[k]$ in which a thread blocks other threads from gaining access to object d , i.e., the period between the event $\langle\sigma_i(d, thread[n])|\sigma_i(s, thread[n])\rangle$ in which $thread[n]$ gains access to object d, and the event $\phi_i(d, thread[n])$ in which $thread[n]$ releases d (Figure 4). Note that each of $c[s, d]$'s three elements is a random variable (for which maintaining the first three moments provides sufficient accuracy).

Pairwise request probabilities. When estimating the pairwise-state $c[s, d]$, we use the *(pairwise) request probability*, $R(s, d)$, of events that are related to $\mathscr{G}$'s edge (s, d) to occur. When given the history of system events, we define the probability of the conditional demand event $\delta(d|s)$ to occur immediacy after the supply event $\sigma(s)$. We also consider the case in which the system does not have access to this information. In that case, we estimate $R(s, d)$ while assuming that $\delta(d|s)$ occurrence depends only on the system parameters, i.e., N, M, $\{job_i\}_{i\in[1,J]}$ and $\{\lambda_{i,n}\}_{i\in[1,J],n\in[1,N]}$, rather than requiring the availability of the event history.

Figure 4: (s, d)'s delay $s.D[k]$ and blocking $s.B[k]$ start from $\delta(d|s)$, and respectively, $\sigma(d|s)$, and both end at $\phi(d|s)$, $d = object[k]$

For a randomly chosen work cycle that includes the event, $\sigma(s)$, of a thread gaining access to item s, we define $\Omega(s) = \{\delta(d|s) : (s, d) \in E$ is an edge in $\mathscr{G}\} \cup \{\phi(s|s)\}$ as the probability space of the possible events to occur immediately after $\sigma(s)$. Moreover, $R(s, d)$ and $R(s, s)$ are the probabilities of $\Omega(s)$'s events $\delta(d|s)$, and respectively, $\phi(s|s)$. Namely, $R(s, d)$ denotes the probability of a demand event, $\delta_\bullet(thread[\bullet], d)$ to occur immediately after the supply event, $\sigma_\bullet(s, thread[\bullet])$ and in the same (randomly chosen) work cycle. Moreover, $R(s, s)$ denotes the probability of a release event, $\phi_\bullet(s, thread[\bullet])$, to occur immediately after its related supply event, $\sigma_\bullet(s, \; thread[\bullet])$ and during the same (randomly chosen) work cycle. Note that $R(s, s)$'s definition requires that s is the last object for the thread to gain access to during that work cycle.

Since $R(s, d)$ and $R(s, s)$ depend on the history of events, we further detail their definitions by using the notations $R_t(s, d)$ and $R_t(s, s)$. We restrict the (random) choice of the work cycle to the time interval $t = [t_{start}, t_{end}]$ and assume the awareness of all events that occurred in the system
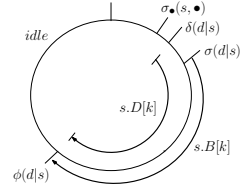
during that period. For the time interval $t = [t_{start}, t_{end}]$, we define $R_t(s,d) = \alpha_t(s,d)/\eta_t(s)$, to be the number of $\delta(d|s)$ occurrences, over the number of $\sigma(s)$ occurrences and $R_t(s,s) = \beta_t(s)/\eta_t(s)$ to be the number of $\phi(s|s)$ occurrences, over the number of $\sigma(s)$ occurrences, where $\#_t X$ denotes the number of event X occurrences in $t = [t_{start}, t_{end}]$. Moreover, $\alpha_t(s,d) = \Sigma_{i,n} \#_t \delta_{i,n}(d|s)$, when $s = object[j] : j \in [1, M-1]$, and $\alpha_t(s,d) = \Sigma_i \#_t \delta_{i,n}(d|s)$, when $s = thread[n] : n \in [1, N]$, the number of $\delta(d|s)$ events that occurred during $t = [t_{start}, t_{end}]$. Furthermore, $\beta_t(s) = \Sigma_{i,n} \#_t \phi_{i,n}(s|s)$ and $\eta_t(s) = \Sigma_{i,n} \#_t \sigma_i(s, thread[n])$ denote the number of $\phi(s|s)$, and respectively, $\sigma(s)$ events that occurred during $t = [t_{start}, t_{end}]$. Note that profiling tools can be the basis for estimating $R_t(s,d)$ and $R_t(s,s)$. We also propose an estimation of $R(s,d)$ and $R(s,d)$ (Section 5) for the case in which these probabilities depend only on N, M, $\{job_i\}_{i \in [1,J]}$ and $\{\lambda_{i,n}\}_{i \in [1,J], n \in [1,N]}$ (and thus t's history of events is not required to be available).

We define the *request probability matrix R* to be a $(N + M) \times (N + M)$ row stochastic matrix. The matrix R has a block form, where $R_{N,N}$ is an $N \times N$ zero matrix, $R_{N,M} = (R(thread[n], object[j]))_{n \in [1,N], j \in [1,J]}$ is an $(N \times M)$ matrix, $R_{M,N}$ is an $M \times N$ zero matrix and $R_{M,M} = (R(object[j], object[k]))_{j,k \in [1,M]}$ is an (upper triangular) $M \times M$ matrix, i.e.,

$$R = \left(\begin{array}{c|c} R_{N,N} & R_{N,M} \\ \hline R_{M,N} & R_{M,M} \end{array} \right) \tag{1}$$

2.7 Item inter-demand period

We refer to $object[k]$'s *inter-demand period*, $\mathcal{T}_{object[k]}$, as the period between two consecutive conditional (demand) events for accessing an object in $\{object[\ell] : \ell \in (k, M)\}$ immediately after gaining access to item $object[k]$ by two, possibly different, threads, where $k \in [1, M-1]$. Namely, $\mathcal{T}_{object[k]}$ is the period between $\delta(object[j]|object[k])$ and the successive conditional event $\delta(object[j']|object[k])$, where $j, j' \in (k, M)$. We refer to $thread[n]$'s *inter-demand period*, $\mathcal{T}_{thread[n]}$ as the period between $\delta_i(thread[n], object[i_1])$ and the demand event $\delta_{i'}(thread[n], object[i'_1])$, where $i, i' \in [1, J]$ and $object[i_1]$, $object[i'_1]$ are the first objects in the object vectors of job_i, and respectively, $job_{i'}$, where $thread[n]$ carries out job_i and $job_{i'}$ consecutively. For example, we can use $1/E(\mathcal{T}_{thread[n]})$ for estimating the number of job completions of $thread[n]$ per time unit, where $E(\mathcal{T}_s)$ is $\mathcal{T}_s$'s expected value. In case $s = object[d]$, $1/E(\mathcal{T}_{object[d]})$ estimates the number of job completions that include $object[d]$ per time unit.

We refer to $\kappa_{i,n} = c_{i,n}/\mathcal{T}_{thread[n]}$ as job_i's *completion rate* on $thread[n]$, which is the fraction of $thread[n]$'s inter-demand period, $\mathcal{T}_{thread[n]}$, due to job_i, where $i \in [1, J]$, $n \in [1, N]$. We use $c_{i,n}$ as a constant for which $\forall n \in [1, N] : \Sigma_{i=1}^{J} c_{i,n} = 1$, and the weights $c_{i,n}$ are a function of the job arrival rates to $thread[n]$, for example $c_{i,n} = \lambda_{i,n} / (\Sigma_{i=1}^{J} \lambda_{i,n})$.

2.8 Shared-object system equilibria

For a given system, $\psi(\mathcal{G}) = \{c[s,d]\}_{(s,d) \in E}$ is the *system state (set)*, where $\mathcal{G} = (V, E)$ is the acquisition graph. For a given $\psi(\mathcal{G})$, the set $\tau(\mathcal{G}) = \{\mathcal{T}_{item}\}_{item \in V \setminus \{object[M]\}}$ is the *inter-demand period of the system*.

Suppose that a system is in a state in which the job arrival rates are equal to the job completion rates, i.e., $\forall i \in [1, J] \; \forall n \in [1, N]$, $\lambda_{i,n} = \kappa_{i,n}$. We say that $\psi^*(\mathcal{G}) = \{c^*[s,d]\}_{(s,d) \in E}$, is the *shared-Object System Equilibrium* (OSE). Given $\psi^*(\mathcal{G})$, the respective inter-demand period of the system is $\tau^*(\mathcal{G}) = \{\mathcal{T}^*_{item}\}_{item \in V \setminus \{object[M]\}}$. For a given $\varepsilon > 0$ and an OSE $\psi^*(\mathcal{G})$, we say that the system state $\psi(\mathcal{G})$ is an *ε-OSE* when $\forall item \in V \setminus \{object[M]\}, \mathcal{T}^*_{item} \in \tau^*(\mathcal{G}), \mathcal{T}_{item} \in \tau(\mathcal{G})$:

Estimating $R(s,d)$ **(Section 5)** We define the following characteristic functions with values in $\{0,1\}$: $starts_i(\langle s_1, ..., s_\ell\rangle) \Leftrightarrow objs_i = (s_1, ..., s_\ell, \bullet)$, $includes_i(\langle s_1, ..., s_\ell\rangle) \Leftrightarrow objs_i = (\bullet, s_1, ..., s_\ell, \bullet)$ and $ends_i(\langle s_1, ..., s_\ell\rangle) \Leftrightarrow objs_i = (\bullet, s_1, ..., s_\ell)$, where $\langle s_1,...,s_\ell\rangle$ is a vector of objects, $objs_i$ is the object vector of job_i and $i \in [1,J]$. When $s = thread[n] : n \in [1,N]$ and $k \in \{1,2\}$, $(\Sigma_i\lambda_{i,n} \cdot starts_i(\langle k\rangle))/(\Sigma_i\lambda_{i,n})$, and respectively, $(\Sigma_i\lambda_{i,n} \cdot includes_i(\langle 1,2\rangle))(\Sigma_i\lambda_{i,n} \cdot includes_i(\langle 1\rangle))$ approximate $R(s,d)$. Moreover, $\forall k \in \{1,2\}$: $(\Sigma_{i,n}\lambda_{i,n} \cdot ends_i(\langle k\rangle))(\Sigma_{i,n}\lambda_{i,n} \cdot includes_i(\langle k\rangle))$ approximates $R(s,s)$.

Estimating $s.B[k]$ **(Section 6)** Let $s \in \{thread[n], object[1]\}$. Observe that the blocking periods $s.B[2]$ are only due to paths that finish in $object[2]$ and thus there are no dependencies for their estimation, i.e., $s.B[2] = A + R(s, object[2]) \cdot R(object[2], object[2]) \cdot f_{s,object[2]}$. On the contrary, the blocking time of a thread's demand to $object[1]$, say $thread[n].B[1]$, depends on the possibility of $thread[n]$ to demand $object[2]$ and the respective delay, $object[1].D[2]$. That is, $thread[n].B[1] = A + R(thread[n], object[1]) \cdot R(object[1], object[1]) \cdot f_{thread[n],object[1]} + R(thread[n], object[1]) \cdot R(object[1], object[2]) \cdot object[1].D[2]$, where $f_{s,d}$ is the average of job operation times for jobs with paths $(\bullet, s, d)$ weighed by the probability, $R(s,d) \cdot R(d,d)$, of such events to occur.

Estimating $\mathcal{T}_d$ **(Section 7)** Let $arrivals(d)$ be the set of items s, such that a conditional demand event $\delta(d|s)$ can occur. Object d's inter-demand period is $\mathcal{T}_d = \Sigma_{s\in arrivals(d)}(\omega(s, d) \cdot (\mathcal{T}_s + A + R(s,d) \cdot R(d,d) \cdot f_{s,d}))/\Sigma_{s\in arrivals(d)} \omega(s, d)$, where the weight is $\omega(s, object[k]) = R(s, object[k]) \cdot s.T[k]$, $arrivals(object[1]) = thread$, while $arrivals(object[2]) = thread \cup \{object[1]\}$ and $thread = (thread[1], ..., thread[N])$.

Figure 5: Estimating $R(s,d)$, $s.B[k]$ and $\mathcal{T}_d$ when $M = 2$.

$|\mathcal{T}_{item}^* - \mathcal{T}_{item}| < \varepsilon$. Namely, the corresponding values of each item in $\tau(\mathcal{G})$ and $\tau^*(\mathcal{G})$ differ by less than ε.

Note that a system cannot always reach a state that satisfies the OSE conditions, and therefore an ε-OSE. Equilibria are *unreachable* when there is an item with a blocking period that is longer than (or equal to) the inter-arrival time of demand events to that item. For example, when the inter-arrival time of object requests is less or equal to the blocking period of that object. Note that in that case, the item's queue is increasing continuously.

3 The Solution Outline

We consider the case in which the job arrival rates can become equal to the job completion rates. We study how the system satisfies the OSE conditions both in exact and approximated manners. We propose a procedure for finding the approximated equilibria, i.e., ε-OSEs. This procedure considers $\mathcal{G}$'s paths, $(\bullet, s, d, \bullet)$, where s is a system item, $d = object[k]$ and $k \in [1,M]$.

3.1 Estimating $c[s,d]$ and $R(s,d)$

We illustrate a solution for the case of $M = 2$ objects and N threads (Figure 5) and outline the general case solution.

The pairwise request probabilities. The pairwise state, $c[s, d]$, and request probability, $R(s, d)$, are related to the conditional events, $\delta(d|s)$. When estimating the value of the pairwise state, we first need to estimate the probability for $\delta(d|s)$ to occur. Our approach considers both the case in which $R(s, d)$ and $R(s, s)$ are given and the case in which they depend only on the system parameters (Section 2.6), i.e., N, M, $\{job_i\}_{i \in [1, J]}$ and $\{\lambda_{i, n}\}_{i \in [1, J], n \in [1, N]}$. Using the latter assumption, we estimate $R(s, d)$ by the sum of job arrival rates for which threads demand access to d after the supply of item s divided by the sum of job arrival rates due to which supply events for item s occur (Figure 5). Moreover, we estimate the probability $R(s, s)$ by the sum of job arrival rates such that their object vectors finish with s divided by the sum of job arrival rates due to which supply events for item s occur.

The thread's blocking periods. Given the pairwise request probabilities, R, we estimate the blocking period, $s.B[k]$ (Section 6), where $d = object[k]$. Item d's blocking period depends in a recursive manner on the delay, cf. $thread[n].B[1]$'s dependency on $object[1].D[2]$ in Figure 5, which in the general case appears as $s.B[k]$'s dependency on $object[k].D[k'] : k' \in (k, M]$ in Lemma 4. This dependency considers the delay that the thread experiences when further acquiring the remaining objects t on the path $(\bullet, s, d, t, \bullet)$, which the thread needs for completing its job, where $t = object[\ell]$ and $\ell \in (k, M]$. Moreover, $s.B[k]$ depends on the *job completion period*, $f_{s,d}$, due to jobs with paths $(\bullet, s, d)$. We refer to (s, d)'s (job) completion period, $f_{s,d}$, as the time between the event $\sigma(d|s)$, in which a thread gains access to object d (after acquiring item s), which is the job's last object, and the event $\phi(d|s)$, in which that thread, immediately after executing the job operation, releases all objects (including d) that it had acquired during the work cycle that includes this two events (e.g., $f_{item[i_{k-1}], item[i_k]}$ is the period between σ_{i_k} and Φ_i, see Figure 1). We detail the exact way in which such *forward dependencies* exist while the system satisfies the OSE condition (Section 6).

The item inter-demand period. An item-level balance also exists and it is similar to the one that the system keeps for its threads (when satisfying the OSE conditions). Namely, the incoming rate of requests (demands) to access object d, has to balance with the inter-demand period, $\mathcal{T}_d$, which is the time between two consecutive $\delta(x|d)$ and $\delta(y|d)$ events (where x and y are two, possibly different, objects). Note that $\mathcal{T}_d$ depends on the rate of requests (demands) to access d due to jobs with $(\bullet, d, d', \bullet)$ paths, as well as, $(\bullet, d)$ paths. Thus, we estimate $\mathcal{T}_d$, as the sum of the inter-demand period of every item s, $\mathcal{T}_s$, and the job completion period $A + R(s, d) \cdot R(d, d) \cdot f_{s,d}$, times a weight that depends on the pairwise inter-demand period $s.T[k]$ plus the request probability $R(s, d)$, for every item s, such that $(s, d = object[k])$ is an edge in $\mathscr{G}$. Note that d's inter-demand period, $\mathcal{T}_d$, has a *backward dependency*, i.e., it depends on the inter-demand periods $s.T[k]$ and $\mathcal{T}_s$, where (s, d) is an edge in $\mathscr{G}$ and $d = object[k]$. We show the exact manner in which the system maintains this balance in Figure 5 for the case of $M = 2$ and in Section 7 for the general case.

3.2 Resolving interdependencies

Thus far, we have noticed several dependencies. For example, there are forward dependencies in which (s, d)'s blocking period, $s.B[k]$, depends on t's delay, where $d = object[k]$, $t = object[\ell]$ and $\ell \in (k, M]$. Moreover, there is a backward dependency in which (s, d)'s inter-demand period, $\mathcal{T}_d$, depends on the summation of pairwise inter-demand period, $s.T[k]$, for any path $(\bullet, s, d, \bullet)$. Note that more dependencies exist. The definition of the pairwise state $c[s, d]$, implies, for example, that (s, d)'s inter-demand period, $s.T[k]$, depends on s's inter-demand period. Moreover, d's delay depends on its blocking, $s.B[k]$, and the (s, d)'s inter-demand period, $s.T[k]$. Note that these pairwise state variables are inter-dependent due to blocking. We show a way to resolve these

interdependencies by representing the thread work cycles as a subsystem in a way that is not subject to blocking and yet preserves the interdependencies that are related to the paths $(\bullet, s, d, \bullet)$ (Section 8). This approach for resolving forward and backward interdependencies is the basis of the proposed procedure for finding approximate equilibria.

3.3 Finding approximate equilibria

We compute an approximated equilibrium, ε-OSE, when such is reachable. We propose a procedure that always halts (Algorithm 1 presents the solution sketch and we detail the entire procedure in Section 9). It returns the system in an ε-OSE state whenever the job arrival and completion rates become equal, or indicates that the system cannot be in a state of an OSE.

The procedure starts with a system state that represents the case in which all queues are empty (line 4). It then estimates the state of a system in which threads can block one another, and the delay grows as more requests are pending in the queues. The procedure works in iterations and decides when to stop using the system inter-demand period, $\{\mathcal{T}_{item}\}_{item \in V \setminus \{object[M]\}}$, i.e., it stops whenever there is no $item \in V \setminus \{object[M]\}$ for which the change in $\mathcal{T}_{item}$ is greater than ε since the previous iteration (lines 6 to 16).

The procedure repeatedly improves an ε-OSE estimation until the system state satisfies the conditions of an approximated equilibrium. It deals with interdependencies using alternating backward and forward iterations (lines 7, and respectively, 12). Namely, we resolve the forward dependencies in which $(s, object[k])$'s blocking period, $s.B[k]$, depends on $object[\ell]$'s delay by *iterating backwards*, where $\ell \in (k, M]$. This backward iteration starts from $k = M$ and counts downwards. Moreover, it can estimate $s.B[k]$ (in a system that its state satisfies the equilibrium conditions), because all of $(s, object[k])$'s forward dependencies can be resolved. Similarly, we can use *forward iterations* for resolving backward dependencies with respect to d's item inter-demand period, $\mathcal{T}_{object[k]}$, because all of $object[k]$'s backward dependencies are resolved.

This loop also updates the thread inter-demand periods, i.e., the time between job completions (line 11), and exits when no item's inter-demand period changes by at least ε between every two iterations (line 16). Together with the estimation of $\mathcal{T}_{thread[n]}$, the procedure checks whether the OSE condition is violated (Section 7), i.e., if the arrival rate, $\Sigma_{i=1}^{J} \lambda_{i,n}$, of jobs to $thread[n]$, is greater or equal than $1/blocking(n)$, where $blocking(n) = A + \Sigma_{k=1}^{M} R(thread[n], object[k]) \cdot thread[n].D[k]$ is the average time it takes to complete a job for $thread[n]$, and $1/blocking(n)$ is respective rate for $blocking(n)$. In case the OSE condition is violated, the loop breaks and the procedure returns. Each iteration takes $O(M^2 \cdot N^4 + M^3)$ time (see Section 9.5).

4 Background knowledge

Our solution uses tools from queueing networks [5]. Although queueing theory celebrated results provide closed forms for single queues, e.g., M/M/c, M/G/1 [1], and queueing networks, e.g., BCMP [3], Gordon-Newell [9], closed form results are far from been the common case. Specifically, there are no relevant closed-form results that can be used for systems like ours in which a thread can block other threads for a non-exponentially distributed period. Ramesh and Perros [18] consider a message passing system of multi-tier server networks in which processes communicate iteratively via what is known in the system community as synchronous I/O (and sometimes called blocking I/O). Our solution requires resolving interdependencies. We use the thread work cycle for showing that our subsystems (Section 8) can represent these interdependencies. We then show that Ramesh-Perros

---

Algorithm 1: Finding an ε-OSE (procedure sketch)

---

1 Input: M, N, $\{job_i\}_{i \in [1,J]}$, $\{\lambda_{i,n}\}_{n \in [1,N], i \in [1,J]}$, R;

2 Output: $(thread[1, N], object[1, M-1], \{\mathcal{T}_v\}_{v:v \neq object[M]})$;

3 Macro: $\mathscr{S}(k) = \{thread\} \cup \{\{object[\ell]\} | \ell \in [1, k-1]\}$;

4 Start by supposing that all queues are empty;

5 repeat

6   let $prevSet \leftarrow$ item inter-demand periods;

7   **for** $k = M$ **to** 1 ($*$ *backward iteration* $*$) **do**

8   **foreach** $\forall S_k \in \mathscr{S}(k), \forall s \in S_k$ **do**

9   Estimate $s.T[k]$, $s.D[k]$ by resolving the $(s, object[k])$-subsystem (Section 8);

10   Estimate $s.B[k]$ (Section 6);

11   **foreach** $n \in [1, N]$ **do** Estimate $\mathcal{T}_{thread[n]}$ and upon OSE condition violation call **return**('no OSE');

12   **for** $k = 1$ **to** $M - 1$ ($*$ *forward iteration* $*$) **do**

13   **foreach** $\forall S_k \in \mathscr{S}(k), \forall s \in S_k$ **do**

14   Estimate $s.T[k]$, $s.D[k]$ by resolving the $(s, object[k])$-subsystem (Section 8);

15   Estimate $\mathcal{T}_{object[k]}$ (Section 7);

16 until *the system has reached equilibrium (test for ε-OSE using the inter-demand period of every item)*;

17 return $(thread[1, N], object[1, M-1], \{\mathcal{T}_v\}_{v:v \neq object[M]})$;

---

subsystems [18] can analyze our subsystems and resolve their interdependencies iteratively. We find ε-OSEs in a similar manner. Namely, we use a framework proposed by Baynat and Dallery [4] for estimating the system state, in a similar manner to Ramesh-Perros [18]. The authors of [18, 4] demonstrate the convergence of their iterative methods via numerical experiments. Baynat and Dallery [4] show that each iteration has polynomial running time, which is $O(M \cdot N^4)$ for the OSE case (Lemma 9).

In the remainder of this section we present the stochastic process through which we calculate the probability of a thread to be idle, using the job arrival process and the time that it takes a thread to complete an arbitrary job (Section 4.1), as well as, a version of the Baynat-Dallery framework adapted to shared-object systems (Section 4.2).

4.1 Idle thread probability

We analyze the stochastic process in which jobs arrive in a shared-object system and are assigned to a thread, $thread[n]$, whenever a previously assigned job is completed (departure) or the thread is idle. This process is characterized by the inter-arrival time of jobs to $thread[n]$, I_n and the respective blocking period B_n, during which $thread[n]$ carries out the job. We explain how an existing method can help us to obtain the probability of $thread[n]$ to be idle, u_n (unoccupied).

We define a Markov chain that has the structure of a quasi-birth-death process (QBD) [8], using I_n and B_n. We consider general distributions of arrival I_n and departure B_n stochastic processes, to which we match Coxian-2 distributions, using their first three moments [2]. We consider (Section 2) the arrival process of job_i to $thread[n]$ to follow an exponential distribution $I[i, n] \sim Exp(\lambda_{i,n})$, as in [7]. Therefore, we can assume that I_n follow an exponential distribution with parameter $\Sigma_{i=1}^{J} \lambda_{i,n}$, where J is the number of the system's jobs (as in [8]). Moreover, we obtain the first three

moments of B_n by the moments of the *thread*[n]'s inter-demand period, $\mathcal{T}_{thread[n]}$, that we calculate in Section 7. By applying moment matching to the first three moments of I_n and B_n, we obtain the arrival, and respectively, departure rates of the continuous-time Markov chain. Thus, we can define a quasi-birth-death (QBD) process that considers the growth and decrease in the number of pending jobs to be assigned to *thread*[n], where each state determines the arrival and departure state of the respective Coxian-2 distribution.

We apply the Matrix Geometric Method (MGM) [16] to find the steady-state probabilities for each state of *thread*[n], due to the Markov chain's QBD structure. We then obtain u_n directly since it equals the sum of the QBD process' probabilities for having no pending jobs, i.e., $u_n = u_{n,1} + u_{n,2}$, where $u_{i,n}$ is the steady-state probability of having no pending jobs and the arrival of a new job being while on phase i of the Coxian-2 distribution, $i \in \{1, 2\}$. The MGM is an iterative method and its running time is in $O(I_{MGM} \cdot m^3)$, where m is the maximum number of states in each QBD level and I_{MGM} is the number of iterations until the method converges. Since for our purposes, m is a constant, e.g., $m = 4$ in the modeling described here, I_{MGM} is a small number and the method converges quadratically, we assume that the running time of the MGM is (practically) constant. We base our assumption that I_{MGM} is a small number on an example that Latouche and Ramaswami [13] give. They argue that in practical settings, I_{MGM} is a small number, considering the case of $I_{MGM} > 40$. Here, $J > 10^{12}$, where J is the number of jobs.

4.2 Baynat-Dallery framework

As a background knowledge, we discuss a variation on Baynat and Dallery's framework [4] that we adapt to the context of shared-object systems (Algorithm 2). The $BDF()$ function denotes our adapted version of Baynat and Dallery's framework. For every $s \in S_k$, this function takes a contention subsystem, $\mathcal{CS}(S_k, k)$, which is the tuple $(\mathcal{H}(S_k, k), (\mathcal{R}_s)_{s \in S_k}, (\mathcal{B}_s)_{s \in S_k})$, as an input and returns an estimation of the delay $s.D[k]$ and inter-demand period $s.T[k]$. Namely, $(s.T[k], s.D[k])_{s \in S_k} = BDF(\mathcal{CS}(S_k, k))$. The solution of Baynat and Dallery is based on iterative approximations of the demand arrival rates and request completion rate to *object*[k] with the ones in the subgraph $\mathcal{H}_s(S_k, k) = (\mathcal{V}_s, \mathcal{E}_s)$, and vice versa, until, for every $s \in S_k$, their absolute difference is below a given threshold.

We complete this section with a detailed explanation of Algorithm 2. The procedure starts by an initialization phase (lines 7–9), which is followed by a repeat-until loop (lines 10–25) and the output calculation (lines 26–27) before returning the output (line 28).

Variables. Lemma 7 of Appendix 8.3 shows that a contention subsystem (Section 8.1) represents the dependencies among the threads in a shared-object system with respect to its state. Let $s \in S_k$ denote a thread, if $S_k = thread$, or an object, if $S_k = \{object[j]\}$, where $j \in [1, k-1]$. For item $v \in \mathcal{V}_s$, we define $\mathcal{I}_s(v)$, $\mathcal{B}_s(v)$ and $\mathcal{C}_s(v)$ to be item v's the inter-arrival time $\mathcal{I}_s(v)$, blocking period $\mathcal{B}_s(v)$, and respectively, inter-demand period $\mathcal{C}_s(v)$. With respect to the $BDF()$ function, $\mathcal{I}_s(v)$ is the time between two demand events to item v, $\mathcal{B}_s(v)$ is the blocking period of an arbitrary demand event to v, and respectively, $\mathcal{C}_s(v)$ is the time between two release events on v. Note that when s is a thread ($S_k = thread$), $\mathcal{I}_s(s)$ is the time between two object release events by s, $\mathcal{B}_s(s)$ is the time from an object release event until the next job completion by s, and respectively, $\mathcal{C}_s(s)$ is the time between two job completions by s. Baynat and Dallery approximate $\mathcal{I}_s(v)$, $\mathcal{B}_s(v)$ and $\mathcal{C}_s(v)$ using exponential distributions with parameters (rates) $\gamma_s(v)$, $\mu_s(v)$, and respectively, $\nu_s(v)$. Note that these random variables depend only on s and v, due to the definition of the contention subsystem (Section 8.1) and Lemma 7 of Appendix 8.3. The $BDF()$ function estimates $\gamma_s(v)$, $\mu_s(v)$ and $\nu_s(v)$

and uses them to compute $s.D[k]$ and $s.T[k]$ for every $s \in S_k$.

We define the subgraph $\mathcal{H}_s(S_k, k) = (\mathcal{V}_s, \mathcal{E}_s)$ of $\mathcal{H}(S_k, k)$ (Section 8.1). Note that when $S_k = thread$, thread $s = thread[n]$ requests to access only to items in $\mathcal{H}_s(S_k, k)$'s subgraph. Similarly, when $S_k = \{object[\ell]\}$, a thread that has access to $s = object[\ell]$ requests to access only to $\mathcal{H}_s(S_k, k)$'s items, where $\ell \in [1, k-1]$. Baynat and Dallery treat these subgraphs as Gordon-Newell networks [9, 6]. The $BDF()$'s repeat-until loop (lines 10–25) alternates between computing $\gamma_s(v)$, $\mu_s(v)$ and $\nu_s(v)$ for the Gordon-Newell network defined by the subgraph $\mathcal{H}_s(S_k, k)$, for every $s \in S_k$ and the same values for every individual item $v \in \mathcal{V}$. The iterations stop when for two consecutive loops there is no $s \in S_k$ and $v \in \mathcal{V}_s$, such that the values of $\mu_s(v)$ differ more than a given ε (line 25).

Initialization phase. The $BDF()$ function initializes $\mu_s(v)$ with $1/E(\mathcal{B}_s(v))$ (lines 7–8) and computes the steady-state probabilities of a thread to demand or have access to item v, after the supply of item s. It does that through the function $stationary()$, which takes the stochastic matrix $\mathcal{R}_s$ as an input. The function $stationary()$ outputs the steady state vector, $steadyStateProbabilities_s$, which has the size of $|\mathcal{V}_s|$. Moreover, $steadyStateProbabilities_s$ satisfies the equations $\pi \cdot \mathcal{R}_s = \pi$ and $\Sigma_{i=1}^{|\mathcal{V}_s|} \pi_i = 1$ (line 9).

The repeat-until loop. The $BDF()$ function's repeat-until loop calculates $\gamma_s(v)$ (lines 12–17), $\nu_s(v)$ (lines 18–22) and $\mu_s(v)$ (lines 23–24). We calculate the Gordon-Newell normalizing constant (line 13) and the marginal probabilities of Gordon-Newell (lines 14–16). Using these marginal probabilities, we calculate $\gamma_s(v)$ for every $s \in S_k$ and $v \in \mathcal{V}_s$ (line 17).

We find the marginal probability of an item to be idle through the $idleProb()$ function (lines 18–21) and then calculate $\nu_s(v)$ (line 22) for every $s \in Z(v)$, where $Z(v) = \{s \in S_k \mid v \in \mathcal{V}_s\}$ (line 5). The $idleProb()$ function calculates item v's marginal probability to be idle through the underlying Markov chain of a multi-class queue with exponential arrivals ($\gamma_s(v)$ for every $s \in Z(v)$). It also calculates the blocking periods ($\mu_s(v)$ for every $s \in Z(v)$). Note that the queue length is limited by the maximum number of pending demands (N when $S_k = thread$ and 1 when $S_k = \{object[j]\}$, $j \in [1, k-1]$) [4, 5].

The calculation $\mu_s(v)$'s new estimates (lines 23–24) happens before the next iteration. In order to check the convergence condition, each iteration begins with storing in the $oldValues$ variable the last estimations of $\mu_s(v)$ for every $s \in S_k$ and $v \in \mathcal{V}_s$ (line 25).

$BDF()$**'s output.** We estimate $s.D[k]$ through the delay in $object[k]$'s queue in the contention subsystem (line 27). Moreover, we obtain an estimation of the inter-demand period $s.T[k]$ through $1/\gamma_s(object[k])$ (line 27). The $BDF()$ function returns the output in line 28.

5 Request probabilities

The fact that the pairwise request probabilities depend on the arrival rates of the corresponding jobs is the basis of our estimation of $R(s, d)$ (Lemma 1) as in the network approximations [8], where (s, d) is an edge in $\mathcal{G}$. In Lemma 2 we prove that our estimations of $R(s, d)$ and $R(s, s)$ define indeed a probability, i.e., for any item s, $R(s, s) + \Sigma_{d \neq s} R(s, d) = 1$. This implies that the probability matrix R, which contains the estimates of $R(s, d)$ and $R(s, s)$, is a stochastic matrix.

Lemma 1. *Equation 2 and Equation 3 approximate $R(s, d)$, when $s = thread[n] : n \in [1, N]$, and respectively, $s = object[j] : j \in [1, M-1]$, see Figure 5 for the definitions of the characteristic functions $starts()$, $includes()$ and $ends()$. Moreover, for any object s, Equation 4 approximates $R(s, s)$.*

Algorithm 2: The $BDF()$ function for estimating delay and pairwise inter-demand period through a contention subsystem

---

1 **Input:** $\mathcal{CS}(S_k, k) = (\mathcal{H}(S_k, k), (\mathcal{R}_s)_{s \in S_k}, (\mathcal{B}_s)_{s \in S_k});$

2 **Output:** $(s.T[k], s.D[k])_{s \in S_k};$

3 **Macros:**

4 $converged(prev, curr) = (\nexists \mu_s(v) \in prev, \mu'_s(v) \in curr : |\mu_s(v) - \mu'_s(v)| \geq \varepsilon);$

5 $Z(v) = \{s \in S_k | v \in \mathcal{V}_s\};$

6 **begin**

7   **foreach** $s \in S_k$ **do**

8   **foreach** $v \in \mathcal{V}_s$ **do** $\mu_s(v) \leftarrow 1/E(\mathcal{B}_s(v));$

9   **foreach** $s \in S_k$ **do** $(steadyStateProbabilities_s(v))_{v \in \mathcal{V}_s} \leftarrow stationary(\mathcal{R}_s);$

10   **repeat**

11   **let** $oldValues = (\mu_s(v))_{s \in S_k, v \in V};$

12   **foreach** $s \in S_k$ **do**

13 $GordonNewellConstant_s \leftarrow \Sigma_{v \in \Theta(s)} steadyStateProbabilities_s(v)/\mu_s(v);$

14   **foreach** $v \in \mathcal{V}_s$ **do**

15 $subgraphMarginalProbs_{s,v}(1) \leftarrow$
$steadyStateProbabilities_s(v)/(\mu_s(v) \cdot GordonNewellConstant_s);$

16 $subgraphMarginalProbs_{s,v}(0) \leftarrow 1 - subgraphMarginalProbs_{s,v}(1);$

17 $\gamma_s(v) \leftarrow$
$\mu_s(v) \cdot (subgraphMarginalProbs_{s,v}(1)/subgraphMarginalProbs_{s,v}(0));$

18   **foreach** $v \in V$ **do**

19   **foreach** $s \in Z(v)$ **do**

20 $itemMarginalProbs_{s,v}(0) \leftarrow$
$idleProb(v, s, \{\gamma_s(v)\}_{s \in S_k}, \{\mu_s(v)\}_{s \in S_k}, (\mathcal{R}_s)_{s \in S_k});$

21 $itemMarginalProbs_{s,v}(1) \leftarrow 1 - itemMarginalProbs_{s,v}(0);$

22   **foreach** $s \in Z(v)$ **do**
$\nu_s(v) \leftarrow \gamma_s(v)(itemMarginalProbs_{s,v}(0)/itemMarginalProbs_{s,v}(1));$

23   **foreach** $s \in S_k$ **do**

24   **foreach** $v \in \mathcal{V}_s$ **do** $\mu_s(v) \leftarrow \nu_s(v);$

25   **until** $converged(oldValues, \{\mu_s(v)\}_{s \in S_k, v \in \mathcal{V}});$

26   **foreach** $s \in S_k$ **do**

27   **let** $(s.D[k], s.T[k]) =$
$(1/(\mu_s(object[k]) \cdot subgraphMarginalProbs_{s,object[k]}(0)), 1/\gamma_s(object[k]);$

28   **return** $(s.T[k], s.D[k])_{s \in S_k};$

$$R(thread[n], d) \approx \frac{(\Sigma_i \lambda_{i,n} \cdot starts_i(\langle d \rangle))}{(\Sigma_i \lambda_{i,n})} \qquad (2)$$

$$R(object[j], d) \approx \frac{\Sigma_i \lambda_{i,n} \cdot includes_i(\langle object[j], d \rangle)}{\Sigma_i \lambda_{i,n} \cdot includes_i(\langle object[j] \rangle)} \qquad (3)$$

$$R(s,s) \approx \frac{(\Sigma_{i,n}\lambda_{i,n} \cdot ends_i(\langle s \rangle))}{(\Sigma_{i,n}\lambda_{i,n} \cdot includes_i(\langle s \rangle))} \tag{4}$$

Equation 2 estimates $R(thread[n], d)$ by the sum of arrival rates of jobs that start with object d to $thread[n]$, divided by the sum of the arrival rates of all the jobs that are assigned to $thread[n]$. Equation 3 estimates $R(s, d)$ by the sum of arrival rates of jobs that include the subvector (s, d) in their object vector (to any thread), divided by the sum of arrival rates of jobs that include the subvector (s) in their object vector (to any thread), where $s = object[j]$ and $j \in [1, M-1]$. Equation 4 estimates $R(s, s)$ by the sum of arrival rates of jobs that their object vector ends with the subvector (s) (to any thread), divided by the sum of arrival rates of jobs that include the subvector (s) in their object vector (to any thread), where $s = object[j]$ and $j \in [1, M]$. Note that in any other case, we define $R(s, d) = 0$, since no conditional demand events $\delta(d|s)$ or conditional release events $\phi(s|s)$ occur in these cases.

We prove that the estimation of the request probabilities in Lemma 1 also defines a probability (Lemma 2).

Lemma 2. *For $R(s, d)$'s estimation (Lemma 1), it holds that: (1) $\Sigma_d R(thread[n], d) = 1$, where $n \in [1, N]$ and d is an object, (2) $R(s, s) + \Sigma_{d \neq s} R(s, d) = 1$, and (3) R is a row stochastic matrix.*

Proof. Equation 5 demonstrates claim (1).

$$
\begin{aligned}
\Sigma_d R(thread[n], d) &= \frac{\Sigma_d \Sigma_i \lambda_{n,i} \cdot starts_i(\langle d \rangle)}{\Sigma_i \lambda_{n,i}} \\
&= \frac{\Sigma_i \lambda_{n,i} \Sigma_d starts_i(\langle d \rangle)}{\Sigma_i \lambda_{n,i}} \\
&= \frac{\Sigma_i \lambda_{n,i} \cdot 1}{\Sigma_i \lambda_{n,i}} \\
&= 1
\end{aligned}
\tag{5}
$$

For any object $s \neq object[M]$, Equation 6 demonstrates claim (2) due to Equation 7, which holds since a job object vector that includes object s, either ends with s or includes more items $d \neq s$.

$$
\begin{aligned}
&R(s,s) + \Sigma_{d \neq s} R(s, d) \\
&= \frac{\Sigma_{n,i}\lambda_{n,i} \cdot ends_i(\langle s \rangle)}{\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s \rangle)} + \frac{\Sigma_{d \neq s}\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s,d \rangle)}{\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s \rangle)} \\
&= \frac{\Sigma_{n,i}\lambda_{n,i} \cdot ends_i(\langle s \rangle) + \Sigma_{d \neq s}\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s,d \rangle)}{\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s \rangle)} \\
&= \frac{\Sigma_{n,i}\lambda_{n,i} \left(ends_i(\langle s \rangle) + \Sigma_{d \neq s}includes_i(\langle s,d \rangle) \right)}{\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s \rangle)} \\
&= \frac{\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s \rangle)}{\Sigma_{n,i}\lambda_{n,i} \cdot includes_i(\langle s \rangle)} \\
&= 1
\end{aligned}
\tag{6}
$$

$$ends_i(\langle s \rangle) + \Sigma_{d \neq s}includes_i(\langle s,d \rangle) = includes_i(\langle s \rangle) \tag{7}$$

15

As for claim (3), we note that claims (1) and (2) imply that our estimation of the block matrices $R_{N,M}$ and $R_{M,M}$ of the matrix R are row stochastic (Equation 1). Since the block matrices $R_{N,N}$ and $R_{M,N}$ are zero matrices, our estimation of R forms a row stochastic matrix.

6 Blocking Periods

We estimate (s,d)'s the blocking period, $s.B[k]$, using the request probabilities and the job completion periods, where $d = object[k]$, $k \in [1, M]$ and $(s,d) \in E$ is an edge in $\mathscr{G}$. This blocking period is an effect of multiple threads' job paths, i.e., the $(\bullet, s, d)$ and the remaining $(\bullet, s, \bullet, d, \bullet)$ paths. The former case corresponds to the job completion period (Lemma 3), whereas the latter depends on the delay of acquiring the path remaining objects (Lemma 4).

Job Completion Periods. In the case of $(\bullet, s, d, \bullet)$ paths, the period for acquiring the remaining objects varies according to (s,d)'s delay, the job completion period depends only on known distributions (the job operation times), and the probability that the related events occur. However, for the case of $(\bullet, s, d)$ paths, $f_{s,d}$ is the operation time average due to jobs with $(\bullet, s, d)$ paths, weighted by the probability for the related events to occur (Lemma 3). This allows us to estimate (s,d)'s blocking time, $s.B[k]$. Lemma 3 defines $f_{s,d} = \Sigma_{i=1}^{J} weight_i(s,d) \cdot O_i$ as a weighted average of all the operation times, where $weight_i(s,d) = (\Sigma_{n=1}^{N} \lambda_{i,n} \cdot ends_i(\langle s,d \rangle)) / W_{s,d}$, if s is an object, $weight_i(s,d) = \lambda_{i,n} \cdot ends_i(\langle s,d \rangle) / W_{s,d}$, if $s = thread[n]$ and $W_{s,d} = \Sigma_{i=1}^{J} weight_i(s,d)$ is the sum of all weights. Note that the weights depend on the arrival rates of jobs with object vectors that end with (s,d), if s is an object and (d), if s is a thread.

Lemma 3. (s,d)'s completion period is a weighted average, $f_{s,d} = \Sigma_{i=1}^{J} weight_i(s,d) \cdot O_i$, of the respective job operation times.

Acquiring the remaining objects. We give an example of how to estimate $s.B[k]$ in a system with two objects (Figure 5). For the general case of M objects, we also have to account for $(\bullet, s, \bullet, r, d, \bullet)$ paths, as in the proof of Lemma 4.

Lemma 4. The (s,d)'s blocking period can be estimated by $s.B[k] = A + R(s, object[k]) \cdot R(object[k], object[k]) \cdot f_{s,object[k]} + \Sigma_{k'=k+1}^{M}(\Pr(\rho = \rho(k')) \cdot object[k].D[k'])$, where $\rho = (\bullet, s, \bullet, object[k], \bullet)$ is a path that includes items s and $d = object[k]$, as well as $\Pr(\rho = \rho(k'))$ is the probability of having $\rho(k') = (\bullet, s, \bullet, object[k], object[k'], \bullet)$ as a path.

Proof. Let $d = object[k]$ and $d' = object[k']$. We define the probability $\Pr(\rho = (\bullet, s, \bullet, d, d', \bullet)) = R(s, d) \cdot R(d, d') + \Sigma_{r=\tilde{s}+1}^{k-1} R(s, object[r]) \cdot [\Sigma_{j=1}^{k-r} R^j(object[r], d)] \cdot R(d, d')$ of a path ρ, where $\tilde{s} = 0$, if $s = thread[n]$, $\tilde{s} = j$, if $s = object[j]$. $R^j(\alpha, \beta)$ is the probability that the path $(\alpha, \bullet, \beta)$ includes $j - 1$ intermediate items between α and β, or equivalently, the probability of an $\mathscr{G}$'s path from α to β to include j edges, which is given by the (α, β) element of the j-th power of the stochastic matrix R. Lemma 4's proof details $s.B[k]$'s forward dependency via its recursive equation that estimates by computing $s.B[M]$ blocking periods, before $s.B[M-1]$ and so on. Moreover, it depends on the delays, $object[k].D[k']$, for acquiring the remaining job objects. We use backward iterations (Section 3) to resolve these dependencies (Section 8).

7 Item inter-demand periods

The item inter-demand period allows us to decide on ε-OSE's condition satisfaction. We estimate the item inter-demand period, which together with the blocking period, $s.B[k]$ (Section 6), are essential for calculating the delay, $s.D[k]$ (Section 8), where $d = object[k]$, $k \in [1, M]$ and $(s, d) \in E$ is an edge in $\mathscr{G}$.

Object inter-demand period. Recall that item d's inter-demand period, $\mathcal{T}_d$, is the time between two demand completions on item d, which in equilibrium equals the time between two demand arrivals to d. Therefore, our inter-demand period estimation, $\mathcal{T}_d$, for an object d depends on (s, d)'s inter-demand periods, which are the time intervals between two consecutive demands to d that first acquire item s, the inter-demand period $s.T[k]$, and the period during which the threads complete such demands.

The estimation of object d's inter-demand period, $\mathcal{T}_d$, is the average of s's inter-demand periods plus the time that demands for $(\bullet, s, d)$ paths block d, where (s, d) is an edge in $\mathscr{G}$. This time of demands is $A + R(s, d) \cdot R(d, d) \cdot f_{s,d}$, weighted by the probability of $(\bullet, s, d, \bullet)$ paths and the (s, d)'s inter-demand period, $s.T[k]$, of demands to d after the supply of item s, due to $(\bullet, s, d, \bullet)$ paths (the weights are normalized by their sum). Namely, the average sums the period between two demand completions of an item s and the job completion time, and the weights incorporate the item dependencies through $s.T[k]$ for all $(\bullet, s, d, \bullet)$ paths and their probability to occur ($d = object[k]$). By generalizing the example system with $M = 2$ objects and N threads (Figure 5) for every $(s, d) \in E$, we obtain Lemma 5.

Lemma 5. *The inter-demand period of $d = object[k]$ is:* $\mathcal{T}_d = (\Sigma_{s \in arrivals(d)} \; \omega(s, \, d) \cdot (\mathcal{T}_s + A + R(s, d) \cdot R(d, d) \cdot f_{s,d})) \, / \, (\Sigma_{s \in arrivals(d)} \; \omega(s, \, d))$*, where* $arrivals(object[k]) = thread \cup \{object[j] \mid j \in [1, k-1]\}$ *and* $\omega(s, \, object[k]) = R(s, \, object[k]) \cdot s.T[k]$.

Lemma 5 details $\mathcal{T}_d$'s backward dependency via its recursive equation that calculates $\mathcal{T}_{object[j]}$, $j \in [1, k-1]$ before calculating $\mathcal{T}_{object[k]}$. Moreover, the calculation of $\mathcal{T}_{thread[n]}$ and $\mathcal{T}_{object[k]}$ depends on the inter-demand periods, which is $thread[n].T[k]$, of $(thread[n], object[k])$ and respectively, $s.T[k]$, where $k \in [1, M]$ and $s \in thread \cup \{object[j] \mid j \in [1, k-1]\}$. We use forward iterations (Section 3) for resolving such dependencies (Section 8).

Thread inter-demand period. We estimate $thread[n]$'s inter-demand period, $\mathcal{T}_{thread[n]}$, by a Markov process that considers both the arrival process of jobs to $thread[n]$ and the time it takes $thread[n]$ to complete a job.

The way that we estimate $\mathcal{T}_{thread[n]}$, uses the $augmntThreadBlock(I(n), blocking(n))$ function. The function input includes the inter-arrival times $I(n)$ and $blocking(n) = A + \Sigma_{k=1}^{M} R(thread[n], object[k]) \cdot thread[n].D[k]$, i.e., the average time it takes $thread[n]$ to complete a job. Moreover, $augmntThreadBlock(I(n), blocking(n))$ outputs the estimation of $\mathcal{T}_{thread[n]}$ and checks if the OSE condition is violated (line 11 in Algorithm 1 of Section 3.3), i.e., the rate $\Sigma_{i=1}^{J} \lambda_{i,n}$ that defines $I(n)$ (we assume $I(n)$ to be exponentially distributed, as in [7]) is greater or equal than $1/blocking(n)$. The function analyzes a queue using Queuing Theory [1]. In our case, we characterize the queue by matching the first three moments of $I(n)$, and respectively, $blocking(n)$ to Coxian-2 distributions (Section 4.1).

Recall that $\mathcal{T}_{thread[n]}$ denotes the period between two consecutive $\delta(d|thread[n])$ and $\delta(d'|thread[n])$ events. The system assigns a job pending to a thread's queue, say $thread[n]$, whenever that thread becomes available, where $n \in [1, N]$. After this assignment (and a random acquisition period of A),
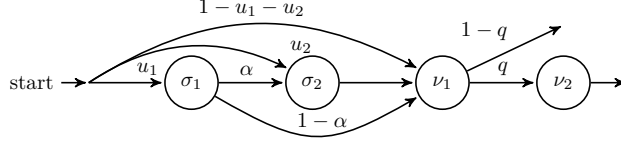
Figure 6: Request generation (Markov) process

the event $\delta(d|thread[n])$ occurs in which the thread requests to access the job's first object; denoted by d.

We estimate $\mathcal{T}_{thread[n]}$ by a Markov chain that depicts both $I(n)$ and $blocking(n)$ (as in Ramesh and Perros [18] but with adaptation to shared-object systems). In Figure 6, σ_1, σ_2 and α define the Coxian-2 job arrival process at $thread[n]$ and ν_1, ν_2 and q define the Coxian-2 of the $thread[n]$'s job completion period ($blocking(n)$), which we obtain by applying moment matching [2] on the first three moments of $I(n)$ and $blocking(n)$. We denote with u_j the probability that upon an arbitrary job completion $thread[n]$ becomes idle and the job arrival process is in phase $j \in \{1,2\}$, as in [18] (Section 4.1). We define $\mathcal{T}_{thread[n]}$ to follow a phase-type distribution (PH) [1], which is defined using the initial probability vector, $c = (u_1, u_2, 1 - u_1 - u_2, 0)$, and a transition matrix Q (Figure 6), where $Q = \left[\begin{array}{c|c} S & S^0 \\ \hline \mathbf{0} & 0 \end{array} \right]$ in its block form, the (x, y) element of S, $S(x, y)$, equals the rate in state x times the transition probability to state y, $x, y \in \{\sigma_1, \sigma_2, \nu_1, \nu_2\}$, and $S^0 = -S\mathbf{1}$. The m-th moment of $\mathcal{T}_{thread[n]}$ is $E[\mathcal{T}_{thread[n]}^m] = (-1)^m m! c S^{-m} \mathbf{1}$ [1].

8 Resolving Dependencies

We showed how to estimate the blocking period on an object, $s.B[k]$, while depending on the delay for acquiring other objects, $object[k].D[k']$ (Section 6), as well as how to estimate the inter-demand periods $\mathcal{T}_{thread[n]}$ and $\mathcal{T}_{object[k]}$, while depending on the $s.T[k]$ pairwise inter-demand periods, due to ($\bullet$, s, $object[k]$, $\bullet$) paths (Section 7). Recall that these variables are inter-dependent due to blocking. Theorem 6 demonstrates that we can resolve these interdependencies by representing the thread work cycles as a subsystem in a way that is not subject to blocking and yet preserves these interdependencies. After the definition of the subsystem, we provide the key steps of the proof by looking into the case of $M = 3$ (Section 8.2). We prove Theorem 6's in Section 8.3 using background knowledge [18] (which we refer to in Section 4).

Theorem 6. *Let $s \in S_k$ and $d = object[k]$, where $S_k \in \{thread\} \cup \{\{object[j]\} \mid j \in [1, k)\}$. The Baynat-Dallery framework can approximate the delay $s.D[k]$ and pairwise inter-demand period $s.T[k]$ through the contention subsystem $\mathcal{CS}(S_k, k) = (\mathcal{H}(S_k, k), (\mathcal{R}_s)_{s \in S_k}, (\mathcal{B}_s)_{s \in S_k})$, where $S_k \in \{thread\} \cup s_k$ and $s_k = \{\{object[i]\} \mid i \in [1, k-1]\}$. The running time of each framework iteration is $O(M \cdot N^4)$.*

8.1 Contention subsystems

Given any pair of system items, s and $d = object[k]$, we calculate the (pairwise) state $c[s, d]$ for $\mathcal{G}$'s edge (s, d) using a construction that we name *contention subsystem* $\mathcal{CS}(S_k, k)$, where $k \in [1, M]$,

$s \in S_k$ and S_k is either the set of all threads, $S_k = thread$, representing $(thread[n], d, \bullet)$ job paths, or a set including a single object, $S_k = \{object[j]\}$, $j \in [1, k-1]$, representing $(\bullet, object[j], d, \bullet)$ paths. For every item $s \in S_k$, we use the thread work cycle, $cycle(thread[n], job_j)$ of $thread[n]$, $n \in [1, N]$, carrying out job_j, (Section 2), to show that the contention subsystem $\mathcal{CS}(S_k, k)$ represents the state of the shared-object system, with respect to the interdependencies among the delay $s.D[k]$ and the pairwise inter-demand period $s.T[k]$ when the blocking time is known.

Let $object[k] \in \{object[1], \ldots, object[M]\}$ as well as $s_k = \{\{object[1]\}, \ldots, \{object[k-1]\}\}$ (i.e., $s_k = \emptyset$ when $k = 1$) and $S_k \in \{thread\} \cup s_k$, where $thread = \{thread[1], \ldots, thread[N]\}$. Moreover, let $Rel(thread, k) = [1, M] \setminus \{k\}$ and $Rel(\{object[i]\}, k) = [i+1, M] \setminus \{k\}$, where $object[i] \in s_k$. We partition the $(\bullet, s, \bullet)$ paths to three sets, $\mathcal{P}(s, d) = \cup_{\ell \in [1,3]} \mathcal{P}_\ell$, where $\mathcal{P}_1 = \{path \mid path = (\bullet, s, d, \bullet)\}$, $\mathcal{P}_2 = \{path \mid path = (\bullet, s, object[i], \bullet, d, \bullet) \wedge i \in Rel(S_k, k) \setminus [k+1, M]\}$, $\mathcal{P}_3 = \{path \mid path = (\bullet, s, \bullet) \wedge d \notin path\}$ and $Rel(thread, k) = [1, M] \setminus \{k\}$.

A *contention subsystem*, is denoted by $\mathcal{CS}(S_k, k) = (\mathcal{H}(S_k, k), (\mathcal{R}_s)_{s \in S_k}, (\mathcal{B}_s)_{s \in S_k})$ and defined by (1)–(3).

(1) The *contention graph* $\mathcal{H}(S_k, k) = (\mathcal{V}, \mathcal{E})$ has the set of vertices $\mathcal{V} = \cup_{s \in S_k} \mathcal{V}_s$, and the set of edges $\mathcal{E} = \cup_{s \in S_k} \mathcal{E}_s$, such that for every $s \in S_k$, $\mathcal{V}_s = \{s\} \cup \{d\} \cup \{relay(s, j) \mid j \in Rel(S_k, k)\}$ and $\mathcal{E}_s = \mathcal{E}_s^1 \cup \mathcal{E}_s^2 \cup \mathcal{E}_s^3$, where $\mathcal{E}_s^1 = \{(s, d), (d, s)\}$, $\mathcal{E}_s^2 = \cup_{j \in Rel(S_k, k) \setminus [k+1, M]} \{(s, relay(s, j)), (relay(s, j), d), (d, s)\}$, and $\mathcal{E}_s^3 = \cup_{j \in Rel(S_k, k)} \{(s, relay(s, j)), (relay(s, j), s)\}$, i.e., $\mathcal{E}_s^k$ corresponds to the partition $\mathcal{P}_k$, where $k \in [1, 3]$. Note that $\mathcal{H}(S_k, k)$ is a simple graph, i.e., there are no multiple edges between two vertices. Moreover, $relay(s, j)$ is a distinct copy of a relay $object[j]$, $j \in Rel(S_k, l)$, for $s \in S_k$.

(2) The *request probability matrices* $\mathcal{R}_s$ for $\mathcal{H}(S_k, k) = (\mathcal{V}, \mathcal{E})$ and $s \in S_k$. $\mathcal{R}_s(s, d)$ depicts the probability of a path $(\bullet, s, d, \bullet)$. Moreover, $\mathcal{R}_s(s, relay(s, j))$ depicts the probability of a path $r = (\bullet, s, object[j], \bullet)$, while $\mathcal{R}_s(relay(s, j), d)$ depicts the probability that $r = (\bullet, s, object[j], \bullet, d, \bullet)$. Furthermore, $\mathcal{R}_s(v, s)$, $v \in \mathcal{V}_s \setminus \{s\}$, depicts the probability of a thread becoming idle or starting a new job after the completion of a job, which is a certain event and therefore $\mathcal{R}_s(v, s) = 1$, $v \in \mathcal{V}_s \setminus \{s\}$.

(3) The *blocking periods*, $(\mathcal{B}_s)_{s \in S_k}$, where $\mathcal{B}_s$ is a function over the set of items in $\mathcal{V}_s$, and $s \in S_k$ refers to the thread blocking periods on each of $\mathcal{V}_s$'s items. Note that E_s^i forms a directed circle in $\mathcal{H}(S_k, k)$, where $s \in S_k$ and $i \in [1, 3]$. A demand request to d that follows the supply of $s \in S_k$ and possibly the supply of a relay object, $object[j]$, $j \in Rel(S_k, k) \setminus [k+1, M]$, is blocking d for $\mathcal{B}_s(d) = s.B[k]$ time. A demand request to server $relay(s, j)$ that follows the supply of $s \in S_k$, blocks that server for a period of $\mathcal{B}_s(relay(s, j)) = s.D[j]$, if $j \in [k+1, M]$ and $\mathcal{B}_s(relay(s, j)) = W(s, relay(s, j), d)$, if $j \in Rel(S_k, k) \setminus [k+1, M]$, where $W(s, relay(s, j), d)$ equals the delay $s.D[j]$ minus the blocking period of a possibly subsequent demand event to $d = object[k]$. Once a job is completed, another demand event follows the supply of $s \in \mathcal{V}_s$ after a period of $\mathcal{B}_s(s) = \mathcal{T}_s$.

8.2   The case of systems with $M = 3$ objects

We use an illustrative example that shows how the contention subsystem of $S_k = thread$ and $d = object[2]$ represents the dependencies among the threads of a system with $M = 3$ objects and N threads, with respect to the delays and pairwise inter-demand periods when the blocking times and the item inter-demand periods are known. We construct the contention subsystem $\mathcal{CS}(thread, 2) = (\mathcal{H}(thread, 2), (\mathcal{R}_s)_{s \in thread}, (\mathcal{B}_s)_{s \in thread})$ based on the work cycles related to the delay $thread[n].D[2]$ and inter-demand period $thread[n].T[2]$, for every $s = thread[n] \in thread$ and $d = object[2]$. We explain the representation of work cycles by a contention graph, which is

(a) Contention graph for $M = 3$ (b) The case of $\mathcal{P}_1$ (c) The case of $\mathcal{P}_2$ (d) The case of $\mathcal{P}_3$
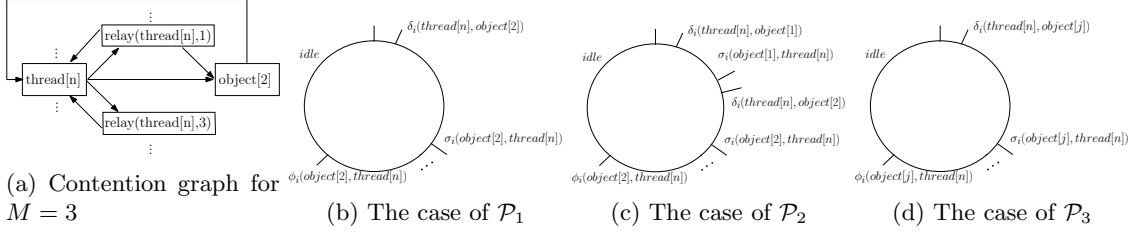
Figure 7: The contention graph for $\mathcal{CS}(thread, 2)$ and the work cycles partitions, $\mathcal{P}(s, d) = \cup_{\ell \in [1,3]} \mathcal{P}_\ell$, of $(s, \bullet)$ paths, where $\mathcal{P}_1 = \{\chi \mid \chi = (s, d, \bullet)\}$, $\mathcal{P}_2 = \{\chi \mid \chi = (s, object[1], d, \bullet)\}$, $\mathcal{P}_3 = \{\chi \mid \chi = (s, \bullet) \wedge d \notin \chi\}$, $s = thread[n]$ and $d = object[2]$.

illustrated in Figure 7a, and the adaptation of the request probabilities and blocking times to the ones of the contention subsystem. The challenge here is to demonstrate that a dynamic system that is based on correlated events with dependencies that are due to blocking and follow non-deterministic schedules can be represented by these subsystems. After demonstrating this part of the proof, the rest of the proof follows by matching between the subsystems presented here to the one by Ramesh-Perros [18], which use a framework proposed by Baynat and Dallery [4] for estimating our system's state.

Contention graph of $\mathcal{CS}(thread, 2)$.   Let $\mathcal{H}(thread, 2) = (\mathcal{V}, \mathcal{E})$ for $S_k = thread$ and $d = object[2]$. (Figure 7a). Given an arbitrary thread, $s = thread[n]$, $n \in [1, N]$, let $\mathcal{P}(s, d) = \cup_{\ell \in [1,3]} \mathcal{P}_\ell$ be a partition of $(s, \bullet)$ paths, where $\mathcal{P}_1 = \{path \mid path = (s, d, \bullet)\}$, $\mathcal{P}_2 = \{path \mid path = (s, object[1], d, \bullet)\}$ and $\mathcal{P}_3 = \{path \mid path = (s, \bullet) \wedge d \notin path\}$. Note that for brevity, we write $s = thread[n]$, $o_1 = object[1]$, $o_3 = object[3]$, and $d = object[2]$ throughout this example, but we clarify this notation when needed. Moreover, let $\mathcal{V} = \cup_{n \in [1,N]} \mathcal{V}_s$ be the union of $\mathcal{V}_s = \{s, relay(s, 1), d, relay(s, 3)\}$, where $relay(s, j)$, $j \in \{1, 3\}$, are s's distinct copies of a relay object, $object[j]$, which allow us to distinguish paths with respect to threads. The contention graph's nodes s, $relay(s, j)$ and $d = object[k]$ represent s, $object[j]$ and, respectively d, in the shared-object system, where $j \in \{1, 3\}$.

The edges $\mathcal{E} = \cup_{n \in [1,N]} \mathcal{E}_s$ follow the path partition cases, $\{\mathcal{E}_\ell\}_{\ell \in [1,3]}$. Let job_i be a job that $s = thread[n]$ carries out. The edge sets $\mathcal{E}_\ell$, where $\ell \in [1, 3]$, are defined as follows.

• $\mathcal{P}_1$'s case refers to work cycles, for which s demands access to d, once it is assigned with job_i. When s gains access to d, job_i might require s to demand access to o_3. Upon job_i's completion s releases any acquired object. Thus, the edges in $\mathcal{E}_1 = \{(s, d), (d, s)\}$ (Figure 7a) represent the work cycle subvectors $(\delta_i(s, d))$, and respectively, $(\sigma_i(d, s), \ldots, \phi_i(d, s), \ldots)$ (Figure 7b).

• $\mathcal{P}_2$'s case refers to work cycles, for which s's demand for access to o_1 is followed by s's demand for access to d after o_1's supply, which is then followed by job_i's completion. Thus, the edges in the set $\mathcal{E}_2 = \{(s, relay(s, 1)), (relay(s, 1), d), (d, s)\}$ (Figure 7a) represent the work cycle subvectors $(\delta_i(s, o_1))$, $(\sigma_i(o_1, s), \delta_i(s, d))$, and respectively, $(\sigma_i(d, s), \ldots, \phi_i(d, s), \ldots)$ (Figure 7c).

• $\mathcal{P}_3$'s case refers to work cycles, for which s demands access to $object[j]$ and then completes job_i, where $j \in \{1, 3\}$. Therefore, the edges in $\mathcal{E}_3 = \{(s, relay(s, j)), (relay(s, j), s)\}$ (Figure 7a), represent the subvector $(\delta_i(s, object[j]))$, and respectively, $(\sigma_i(object[j]), s), \ldots, \phi_i(object[j], s), \ldots)$ of the work cycle (Figure 7d).

$\mathcal{CS}(thread, 2)$**'s blocking times and request probabilities.**   We complete the example in which we show how the contention subsystem represents the dependencies among the threads in

the shared-object system. We refer to an arbitrary job, say job_i, that $s = thread[n]$ carries out and explain how the contention subsystem's request probabilities $(\mathcal{R}_s)_{n \in [1,N]}$ and blocking periods $(\mathcal{B}_s)_{n \in [1,N]}$ represent the request probabilities, and respectively, the blocking periods in the shared-object system. We justify this representation using the work cycle of job_i, when its path is in the path partition $\mathcal{P}_j$, for every $j \in [1,3]$. Note that for each such path partition, the period between two consecutive work cycles completed by s is represented by $\mathcal{B}_s(s) = \mathcal{T}_s$ (Figure 7). Namely, if s carries out job_i and consecutively $job_{i'}$, and their first demand requests are $\delta_i(s, d)$, and respectively, $\delta_{i'}(s, d')$, the blocking period $\mathcal{B}_s(s)$ represents the period between these two events (in the contention subsystem).

The case of $(s, o_1, \bullet)$ paths. Consider the case where s carries out job_i with path $r = (s, o_1, \bullet)$, where $r \in \mathcal{P}_2$, if d is included in r and $r \in \mathcal{P}_3$, if d is not included in r. We present the request probabilities among s, $relay(s, 1)$ and d, as well as the blocking times on each of these items in the contention subsystem.

• *The probability $\mathcal{R}_s(s, relay(s, 1))$.* The probability $\mathcal{R}_s(s, relay(s, 1)) = R(s, o_1)$ (by the definition of R) denotes the contention subsystem event of s demanding access to $relay(s, 1)$, which represents s demanding access to o_1 immediately after job_i's assignment in the shared-object system (figures 7c and 7d, when $j = 1$).

• *The blocking period $\mathcal{B}_s(relay(s, 1))$.* Let $W(s, o_1, d)$ denote the period during which s blocks o_1, minus the possible blocking period on d (after the supply of access to o_1) in the shared-object system. Moreover, let $X(s, o_1, d) = \Pr[(s, o_1, d)] \cdot o_1.B[2]$ denote the (possible) blocking period of $s = thread[n]$ to $d = object[2]$, after gaining access to $o_1 = object[1]$ in the shared-object system, where the probability $\Pr[(s, o_1, d)] = R(s, o_1) \cdot R(o_1, d)$ denotes the event of s demanding access to o_1 and successively to d. Namely, $X(s, o_1, d)$ is the time between the work cycle events $\delta_i(s, d)$ and $\phi_i(d, s)$ times the probability of s to demand access to d after gaining access to o_1 (Figure 7c). Thus, in the shared-object system $W(s, o_1, d) = s.D[1] - X(s, o_1, d)$. Therefore, $\mathcal{B}_s(relay(s, 1)) = W(s, o_1, d)$ represents the period during which s blocks $relay(s, 1)$ before possibly demanding access to d in the contention subsystem.

• *The probabilities $\mathcal{R}_s(relay(s, 1), d)$ and $\mathcal{R}_s(relay(s, 1), s)$.* Let $\mathcal{F}_{(s,1,2)}$ and $\mathcal{F}'_{(s,1,2)}$ denote the events in which s demands access to d after gaining access to o_1 in the shared-object system, and respectively, to $relay(s, 1)$ in the contention subsystem. The event $\mathcal{F}'_{(s,1,2)}$ in the contention subsystem represents the event $\mathcal{F}'_{(s,1,2)}$ in the shared-object system, and therefore, the probability of $\mathcal{F}'_{(s,1,2)}$ is given by $\mathcal{R}_s(relay(s, 1), d) = \Pr[(s, o_1, d)]$ (Figure 7c). Moreover, when the event $\mathcal{F}'_{(s,1,2)}$ (and thus the event $\mathcal{F}_{(s,1,2)}$) does not occur (Figure 7d), the respective job is completed and s becomes idle or starts a new job with probability $\mathcal{R}_s(relay(s, 1), s) = 1 - \mathcal{R}_s(relay(s, 1), d)$.

The case of $(s, d, \bullet)$ paths. Consider the case where job_i's path is $(s, d, \bullet) \in \mathcal{P}_1$. In Figure 7b, s demands access to d immediately after it is assigned with job_i. This is represented in $\mathcal{CS}(thread, 2)$ by s demanding access to d, with probability $\mathcal{R}_s(s, d) = R(s, d)$. Moreover, s blocks d for a period of $\mathcal{B}_s(d) = s.B[2]$ in $\mathcal{CS}(thread, 2)$, which represents the period between the events $\sigma_i(d, s)$ and $\phi_i(d, s)$ in figures 7b and 7c. After the job completion and the release event of d in $\mathcal{CS}(thread, 2)$, s enters, with probability $\mathcal{R}_s(d, s) = 1$, an idle period (of possibly zero length) until it starts carrying out a new job.

The case of (s, o_3) paths. Consider the case where s carries out job_i with path $r = (s, o_3) \in \mathcal{P}_3$. In $\mathcal{CS}(thread, 2)$, s demands access to $relay(s, 3)$, with probability $\mathcal{R}_s(s, relay(s, 3)) = R(s, o_3)$, which represents s demanding access to o_3 immediately after job_i's assignment in the shared-object system (Figure 7d). The blocking period of s on $relay(s, 3)$ is $\mathcal{B}_s(relay(s, 3)) = s.D[3]$, which in the shared-object system represents the time that s is waiting to gain access to o_3 and then

(a) The $\mathcal{H}(S_k, k)$ contention graph, where $[i_1, i_c] = Rel(S_k, k) \setminus [k+1, M]$

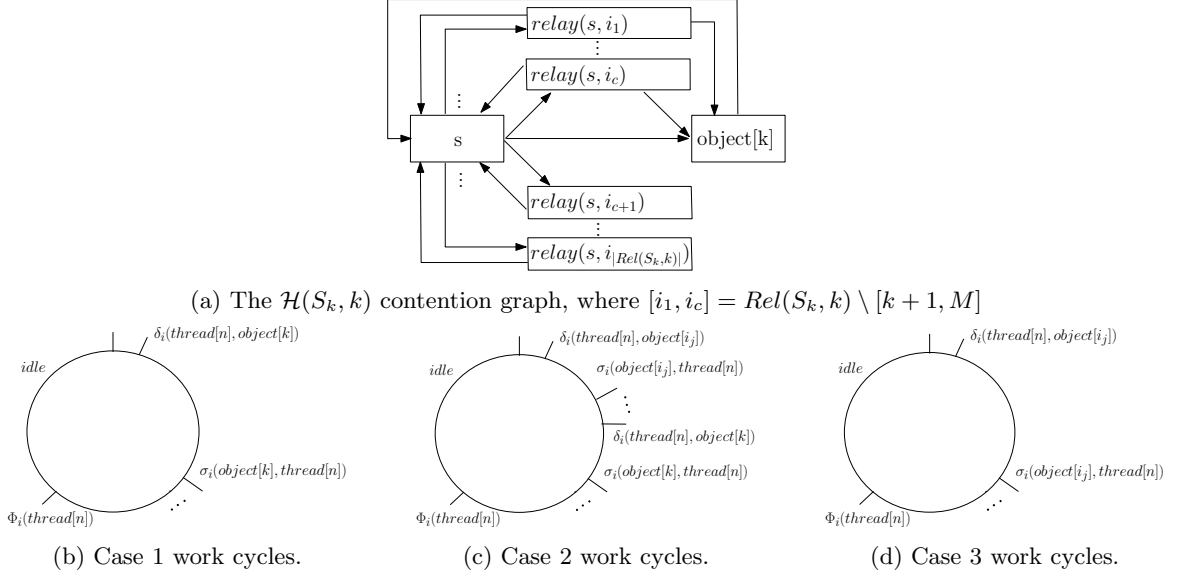(b) Case 1 work cycles. (c) Case 2 work cycles. (d) Case 3 work cycles.

Figure 8: The contention graph for $\mathcal{CS}(S_k, k)$ and the work cycles partitions, $\mathcal{P}(s, d) = \cup_{\ell \in [1,3]} \mathcal{P}_\ell$, of $(\bullet, s, \bullet)$ paths, where $\mathcal{P}_1 = \{path \mid path = (\bullet, s, d, \bullet)\}$, $\mathcal{P}_2 = \{path \mid path = (\bullet, s, object[i], \bullet, d, \bullet) \wedge i \in Rel(S_k, k) \setminus [k+1, M]\}$, $\mathcal{P}_3 = \{path \mid path = (\bullet, s, \bullet) \wedge d \notin path\}$, $Rel(thread, k) = [1, M] \setminus \{k\}$ and $d = object[k]$.

blocking it, i.e., the period between the work cycle events $\delta_i(s, o_3)$ and $\phi_i(o_3, s)$ (Figure 7d). After the job completion and the release event of $relay(s, 3)$ in $\mathcal{CS}(thread, 2)$, s enters, with probability $\mathcal{R}_s(relay(s, 3), s) = 1$, an idle period (of possibly zero length) until it starts carrying out a new job.

The contention subsystem $\mathcal{CS}(thread, d) = (\mathcal{H}(thread, d), (\mathcal{R}_s)_{s \in thread}, (\mathcal{B}_s)_{s \in thread})$, which we described above, represents the dependencies among the thread set, $thread$, and $d = object[2]$ in the shared-object system.

8.3   The case of systems with M objects

In this section we prove that Theorem 6 follows from lemmata 7, 8 and 9 (Corollary 10).

Lemma 7. *Consider a contention subsystem $\mathcal{CS}(S_k, k) = (\mathcal{H}(S_k, k), (\mathcal{R}_s)_{s \in S_k}, (\mathcal{B}_s)_{s \in S_k})$, where $S_k \in \{thread\} \cup s_k$ and $s_k = \{\{object[i]\} \mid i \in [1, k-1]\}$. Suppose that we are given the shared-object system's blocking and item inter-demand periods, as well as the request probabilities R. It holds that $\mathcal{CS}(S_k, k)$ represents the dependencies among the threads in the shared-object system and the system's state.*

Proof. We show a mapping of the shared-object system's state to the contention subsystem $\mathcal{CS}(S_k, k)$. Given the shared-object system's blocking and item inter-demand periods, as well as the request probabilities, we construct the contention subsystem $\mathcal{CS}(S_k, k) = (\mathcal{H}(S_k, k), (\mathcal{R}_s)_{s \in S_k}, (\mathcal{B}_s)_{s \in S_k})$ based on the work cycles related to jobs with $(\bullet, s, \bullet)$ paths, where $s \in S_k$ and $s_k = \{\{object[i]\}$

$| \ i \in [1, k-1]\}$. We explain the representation of work cycles by the contention graph $\mathcal{H}(S_k, k)$, as well as the representation of the shared-object system's request probabilities and state, i.e., blocking, pairwise inter-demand period and delay, by $(\mathcal{R}_s)_{s \in S_k}$, and respectively, $(\mathcal{B}_s)_{s \in S_k}$ in the contention subsystem. This construction is the mapping that proves the lemma's statement.

The proof is organized as follows. In the first part, we construct the contention graph $\mathcal{H}(S_k, k) = (\mathcal{V}, \mathcal{E})$ using the work cycles related to $(\bullet, s, \bullet)$ paths. Moreover, in the second part we show that $(\mathcal{R}_s)_{s \in S_k}$ and $(\mathcal{B}_s)_{s \in S_k}$ represent the dependencies and the shared-object system's state with respect to $(\bullet, s, \bullet)$ paths. In our construction, we assume the knowledge of the item inter-demand periods $\mathcal{T}_v$, the system's state delay and blocking periods, as well as the request probabilities R.

The graph $\mathcal{H}(S_k, k)$ represents the work cycles related to the $(\bullet, s, \bullet)$ paths. Let $\mathcal{H}(S_k, k) = (\mathcal{V}, \mathcal{E})$ be the contention graph of $\mathcal{CS}(S_k, k)$ (Figure 8a) and consider s to be an arbitrary element of S_k. Recall that $Rel(thread, k) = [1, M] \setminus \{k\}$ and $Rel(\{object[\ell]\}, k) = [\ell+1, M] \setminus \{k\}$, where $\ell \in [1, k]$. Moreover, let $\mathcal{P}(s, object[k]) = \cup_{\ell \in [1,3]} \mathcal{P}_\ell$ be a path partition of $(\bullet, s, \bullet)$ paths, where $\mathcal{P}_1 = \{path \mid path = (\bullet, s, object[k], \bullet)\}$, $\mathcal{P}_2 = \{path \mid path = (\bullet, s, object[\ell], \bullet, object[k], \bullet) \wedge \ell \in Rel(S_k, k) \setminus [k+1, M]\}$ and $\mathcal{P}_3 = \{path \mid path = (\bullet, s, \bullet) \wedge object[k] \notin path\}$. We explain the representation of the thread work cycles for jobs with $(\bullet, s, \bullet)$ paths by the graph $\mathcal{H}(S_k, k)$.

We define the elements of $\mathcal{V}$ and $\mathcal{E}$. Let $\mathcal{V} = \cup_{s \in S_k} \mathcal{V}_s$, where $\mathcal{V}_s = \{s, object[k]\} \cup \{relay(s, j) \mid j \in Rel(S_k, k)\}$. The nodes $relay(s, j)$, $j \in Rel(S_k, k)$, are s's distinct copies of a relay object, $object[j]$, which allow us to distinguish paths with respect to threads. The edges $\mathcal{E} = \cup_{s \in S_k} \mathcal{E}_s$ follow the three path partition cases, i.e., $\mathcal{E}_s = \cup_{\ell \in [1,3]} \mathcal{E}_\ell$, where $\mathcal{E}_1 = \{(s, object[k]), (object[k], s)\}$, $\mathcal{E}_2 = \{(s, relay(s, j)), (relay(s, j), object[k]), (object[k], s)\}$ and $\mathcal{E}_3 = \{(s, relay(s, j)), (relay(s, j), s)\}$. Let job_i be a job that $thread[n]$ carries out, such that item s is either included in job_i's object vector or $s = thread[n]$, and let $cycle(thread[n], job_i)$ be the respective work cycle. The edges in the sets $\mathcal{E}_\ell$, where $\ell \in [1, 3]$, represent the events in every possible $cycle(thread[n], job_i)$ after the supply of item s, if s is an object, or after the assignment of job_i to s, if $s = thread[n]$. For brevity, we refer to both events as the supply of item s. The edge sets are defined according to the three sets of the path partition $\mathcal{P}(s, object[k]) = \cup_{\ell \in [1,3]} \mathcal{P}_\ell$ as follows.

- $\mathcal{P}_1$'s case refers to work cycles, for which $thread[n]$ demands access to $object[k]$, immediately after the supply of item s. When $thread[n]$ gains access to $object[k]$, job_i might require $thread[n]$ to demand access to another object, $object[k']$, where $k' \in [k+1, M]$. Upon job_i's completion $thread[n]$ releases any acquired object (event $\Phi_i(thread[n])$ of the work cycle). Thus, the edges $(s, object[k])$ and $(object[k], s)$ of $\mathcal{E}_1$ (Figure 8a) represent the subvector $(\delta_i(thread[n], object[k]))$, and respectively, the subvector $(\sigma_i(object[k], thread[n]), \ldots, \Phi_i(thread[n]))$ of the work cycle (Figure 8b).

- $\mathcal{P}_2$'s case refers to work cycles for which $thread[n]$, after the supply of item s, demands access to $object[j]$, where $j \in Rel(S_k, k) \setminus [k+1, M]$, and subsequently to $object[k]$. Note that, by the definition of $\mathcal{P}_2$, $thread[n]$ might also demand access to other objects in $(object[j], object[k]) \cup [object[k+1], object[M]$. Thus, the edges $(s, relay(s, j))$, $(relay(s, j), object[k])$, $(object[k], s)$ of $\mathcal{E}_2$ (Figure 8a) represent the work cycle subvectors $(\delta_i(thread[n], object[j]))$, $(\sigma_i(object[j], thread[n]), \ldots \delta_i(thread[n], object[k]))$, and respectively, $(\sigma_i(object[k], thread[n]), \ldots, \Phi_i(thread[n]))$ (Figure 8c).

- $\mathcal{P}_3$'s case refers to work cycles, for which $thread[n]$, after the supply of item s, demands access to $object[j]$, where $j \in Rel(S_k, k)$. Note that, by the definition of $\mathcal{P}_3$, $thread[n]$

might also demand access to other objects in ($object[j]$, $object[M]$, but not to $object[k]$. Thus, the edges (s, $relay(s, j)$) and ($relay(s, j)$, s) of $\mathcal{E}_3$ (Figure 8a), represent the subvectors ($\delta_i(thread[n]$, $object[j])$), and respectively, ($\sigma_i(object[j])$, $thread[n]$), ..., $\Phi_i(thread[n])$), where $j \in Rel(S_k, k)$ of the work cycle (Figure 8d).

$(\mathcal{R}_s)_{s \in S_k}$ **and** $(\mathcal{B}_s)_{s \in S_k}$ **represent the dependencies and the shared-object system's state with respect to** ($\bullet$, s, $\bullet$) **paths.** We refer to an arbitrary job, say job_i, that $thread[n]$ carries out and explain how the contention subsystem's request probabilities $(\mathcal{R}_s)_{s \in S_k}$ and blocking periods $(\mathcal{B}_s)_{s \in S_k}$ represent the request probabilities, and respectively, the blocking periods in the shared-object system. We verify this representation using the work cycle of job_i, when its path is in the path partition $\mathcal{P}_j$, for every $j \in [1, 3]$. We remind that throughout this proof we refer to the supply of item s as the supply of access to $object[\ell]$ for $thread[n]$, if $s = object[\ell]$, $\ell \in [1, k-1]$, or the assignment of job_i to $thread[n]$, if $s = thread[n]$.

Note that for each such path partition, the period between two consecutive work cycles for jobs that include item s is represented by $\mathcal{B}_s(s) = \mathcal{T}_s$ (Figure 7). Namely, the blocking period $\mathcal{B}_s(s)$ in the contention subsystem represents the period between the event of $thread[n]$ releasing item s due to job_i, and consecutively, another thread, say $thread[n']$, releasing item s due to a job, say $job_{i'}$, in the shared-object system. We show this representation by looking into three cases of path partitions, i.e., (1) paths ($\bullet$, s, $object[k]$, $\bullet$) in $\mathcal{P}_1$, (2) paths $r = (\bullet$, s, $object[j]$, $\bullet$), where $j \in Rel(S_k, k) \setminus [k+1, M]$ ($r \in \mathcal{P}_2$, if $object[k]$ is included in r and $r \in \mathcal{P}_3$, if $object[k]$ is not included in r) and (3) paths $r = (thread[n]$, $\bullet$, s, $object[j]$, $\bullet$), where $j \in Rel(S_k, k)$, and $object[k] \notin r$, i.e., $r \in \mathcal{P}_3$.

(1) Consider the case where job_i's path is in partition $\mathcal{P}_1$, i.e., $thread[n]$ carries out job_i with path ($\bullet$, s, $object[k]$, $\bullet$). In Figure 8b, $thread[n]$ demands access to $object[k]$ immediately after the supply of item s. This event is represented in the contention subsystem by $thread[n]$ demanding access to $object[k]$, with probability $\mathcal{R}_s(s, object[k]) = R(s, object[k]) / K(S_k, k)$ (by the definition of R), where $K(S_k, k) = \Sigma_{v \in Rel(S_k, k) \cup \{k\}} R(s, object[v])$ is a normalizing constant . Furthermore, $thread[n]$ blocks $object[k]$ for a period of $\mathcal{B}_s(object[k]) = s.B[k]$ in the contention subsystem, which represents the period between the events $\sigma_i(object[k]$, $thread[n])$ and $\Phi_i(thread[n])$ in the work cycles presented in figures 8b and 8c. This is due to the fact that $s.B[k]$ also considers requests for accessing $object[k]$ while following either job path ($thread[n]$, $\bullet$, s, $object[k]$, $\bullet$) or ($thread[n]$, s, $\bullet$, $object[k]$, $\bullet$) (Lemma 4). After the job completion and the release event of $object[k]$, another supply event of item s occurs or it becomes idle. Namely, if $s = thread[n]$, s will either start carrying out the next pending job or it will become idle, and if $s = object[\ell]$, the thread on s's queue top will gain access to s in case the queue is not empty, otherwise s will become idle. These are certain events in the shared-object system and therefore occur with probability $\mathcal{R}_s(object[k], s) = 1$ in the contention subsystem.

(2) Consider the case where $thread[n]$ carries out job_i with path $r = (\bullet$, s, $object[j]$, $\bullet$), where $j \in Rel(S_k, k) \setminus [k+1, M]$. Note that $r \in \mathcal{P}_2$, if $object[k]$ is included in r and $r \in \mathcal{P}_3$, if $object[k]$ is not included in r. We present the request probabilities among s, $relay(s, j)$ and $object[k]$, $j \in Rel(S_k, k) \setminus [k+1, M]$, as well as the blocking times on each of these items in the contention subsystem. We present (i) the probability $\mathcal{R}_s(s, relay(s, j))$, (ii) the blocking period $\mathcal{B}_s(relay(s, j))$ and (iii) the probabilities $\mathcal{R}_s(relay(s, j), object[k])$ and $\mathcal{R}_s(relay(s, j), s)$.

   (i) The probability $\mathcal{R}_s(s, relay(s, j)) = R(s, object[j]) / K(S_k, k)$ (by the definition of

24

R) denotes the contention subsystem event of $thread[n]$ demanding access to $relay(s, j)$ after the supply of item s, which represents $thread[n]$ demanding access to $object[j]$ immediately after the supply of item s in the shared-object system (figures 8c and 8d, when $j \in Rel(S_k, k) \setminus [k+1, M]$).

(ii) Consider the case where $thread[n]$ might demand access to objects in $(object[j], object[M], $ after the supply of $object[j]$ in the shared-object system. Let $W(s, object[j], object[k])$ as the period during which $thread[n]$ blocks $object[j]$ minus $thread[n]$'s possible blocking period on $object[k]$ in the shared-object system, due to a path in $\mathcal{P}_2$. We refer to $X(s, object[j], object[k]) = \Sigma_{\ell=j}^{k-1} \Pr[(s, object[j], \bullet, object[\ell], object[k])] \cdot object[\ell].B[k]$ as the possible blocking period of $thread[n]$ to $object[k]$, after gaining access to $object[j]$ in the shared-object system. Note that the probability $\Pr[(s, object[j], \bullet, object[\ell], object[k])] = R(s, object[j]) \cdot [\Sigma_{k=1}^{j-\ell} R^k(object[\ell], object[j])] \cdot R(object[\ell], object[k])$ denotes the event in which $thread[n]$, after the supply of item s, demands access to $object[j]$, possibly to other objects in $(object[j], object[\ell])$, to $object[\ell]$ and successively to $object[k]$. Namely, let $X(s, object[j], object[k])$ denote the time between the work cycle events $\delta_i(thread[n], object[k])$ and $\Phi_i(thread[n])$ times the probability of $thread[n]$ to demand access to $object[j]$, after gaining access to s, and subsequently to $object[k]$ (Figure 8c). Thus, in the shared-object system $W(s, object[j], object[k]) = s.D[j] - X(s, object[j], object[k])$. Therefore, the period during which $thread[n]$ blocks $relay(s, j)$, after the supply of item s, and before possibly demanding access to $object[k]$ in the contention subsystem is represented by $\mathcal{B}_s(relay(s, j)) = W(s, object[j], object[k])$.

(iii) Let $\mathcal{F}_{(s,j,k)}$ and $\mathcal{F}'_{(s,j,k)}$ denote the event in which $thread[n]$, demands access to $object[j]$, after the supply of item s, and subsequently to $object[k]$ in the shared-object system, and respectively, the event in which $thread[n]$, after the supply of item s, demands access to $relay(s, j)$ and consecutively to $object[k]$ in the contention subsystem. Note that $thread[n]$ might demand access to other objects in $(object[j], object[k])$ in the shared-object system. The event contention subsystem $\mathcal{F}'_{(s,j,k)}$ represents the event $\mathcal{F}'_{(s,j,k)}$ in the shared-object system, and therefore, the probability of $\mathcal{F}'_{(s,j,k)}$ is given by $\mathcal{R}_s(relay(s, j), object[k])$ which equals to $\Pr[(s, object[j], object[k])]$ (Figure 8c). Moreover, when the event $\mathcal{F}'_{(s,j,k)}$ (and thus the event $\mathcal{F}_{(s,j,k)}$) does not occur (Figure 8d), the respective job is completed and $thread[n]$ becomes idle or starts a new job with probability $\mathcal{R}_s(relay(s, j), s) = 1 - \mathcal{R}_s(relay(s, j), object[k])$.

(3) Consider the case where $thread[n]$ carries out job_i with path $r = (thread[n], \bullet, s, object[j], \bullet)$, where $j \in Rel(S_k, k)$, and $object[k] \notin r$, i.e., $r \in \mathcal{P}_3$. In the contention subsystem, $thread[n]$, after the supply of item s, demands access to $relay(s, j)$, with probability $\mathcal{R}_s(s, relay(s, j)) = R(s, object[j])$ (by the definition of R), which represents $thread[n]$ demanding access to $object[j]$ immediately after the supply of item s in the shared-object system (Figure 8d). The blocking period of $thread[n]$ on $relay(s, j)$ is $\mathcal{B}_s(relay(s, j)) = s.D[j]$, which in the shared-object system represents the time that $thread[n]$ is waiting to gain access to $object[j]$, blocks it while it that does read well here. Too many 'it'. Please use the name $thread[n]$ and $object[j]$. Another way to go is to use (conditional) events... the time between $\delta_{i,n}(d|s)$ and .... possibly demands access to other objects (except for $object[k]$) and releases all of its acquired objects. Namely, $\mathcal{B}_s(relay(s, j))$ represents the period between the work cycle events $\delta_i(thread[n], object[j])$ and $\Phi_i(thread[n])$, after the supply of item s (Figure 8d). After the job completion and the release event of $relay(s, j)$ in the contention subsystem, either another thread gains access to item s

or item s becomes idle (no thread is accessing it), i.e., this event is certain to happen in the shared-object system.

The $\mathcal{CS}(S_k,\ k) = (\mathcal{H}(S_k,\ k),\ (\mathcal{R}_s)_{s \in S_k},\ (\mathcal{B}_s)_{s \in S_k})$ contention subsystem, which we described above, represents the dependencies among the item set, S_k, and $object[k]$ in the shared-object system. Therefore, the proof is complete.

Lemma 8. *The framework of Baynat and Dallery can approximate the delay $s.D[k]$ and inter-demand period $s.T[k]$ through the contention subsystem $\mathcal{CS}(S_k, k)$.*

Proof. We first give the definition of a *Ramesh-Perros subsystem* (RPS) that is introduced in [18] and show that a contention subsystem can be directly mapped to an RPS. Ramesh and Perros show that we can find the pairwise inter-demand period, and delay of (blocking) communications in an RPS using a framework proposed by Baynat and Dallery in [4]. Thus, we can use the Baynat-Dallery framework to estimate the values of $s.T[k]$ and $s.D[k]$, for every $s \in S_k$, given a contention subsystem $\mathcal{CS}(S_k, k)$ as an input. In Appendix 4.2, we present an adapted version of the Baynat-Dallery framework (Algorithm 2) and explain the calculation of the delay $s.D[k]$ and inter-demand period $s.T[k]$, given the contention subsystem $\mathcal{CS}(S_k, k)$.

A Ramesh-Perros subsystem is defined as follows. Let $\{o[1],\ \ldots,\ o[M]\}$ be a set of servers, such that tier-k includes only server $o[k]$, where $k \in [1, M]$, and $\{t[1],\ \ldots,\ t[N]\}$ be a set of clients. Moreover, let $s_k = \{\{o[1]\}, \ldots, \{o[k-1]\}\}$ (i.e., $s_k = \emptyset$ when $k = 1$) and $S_k \in \{\{t[1],\ \ldots,\ t[N]\}\} \cup s_k$. A Ramesh-Perros subsystem, is denoted by $\mathcal{RP}(\mathbf{S}_k, k) = (\mathbf{H}(S_k,\ k),\ (\mathbf{R}_x)_{x \in S_k},\ (\mathbf{B}_x)_{x \in S_k})$ and defined as follows:

(1) The *RPS graph* $\mathbf{H}(S_k,\ k) = (\mathbf{V}, \mathbf{E})$ has the set of vertices $\mathbf{V} = \cup_{x \in S_k} \mathbf{V}_x$, and the set of edges $\mathbf{E} = \cup_{x \in S_k} \mathbf{E}_x$, such that $\mathbf{V}_x = \{x\} \cup \{o[k]\} \cup \{relay(x, j) \mid j \in \mathtt{Rel}(S_k,\ k)\}$ and $\mathbf{E}_x = \mathbf{E}_x^1 \cup \mathbf{E}_x^2 \cup \mathbf{E}_x^3$, where $\mathtt{Rel}(\{t[1],\ \ldots,\ t[N]\},\ k) = [1, M] \setminus \{k\}$, $\mathtt{Rel}(\{o[i]\},\ k) = [i+1, M] \setminus \{k\}$, where $o[i] \in s_k$, $\mathbf{E}_x^1 = \{(x,\ o[k]),\ (o[k],\ x)\}$, $\mathbf{E}_x^2 = \cup_{j \in \mathtt{Rel}(S_k, k) \setminus [k+1, M]} \{(x,\ relay(x,\ j)),\ (relay(x, j),\ o[k]),\ (o[k],\ x)\}$, and $\mathbf{E}_x^3 = \cup_{j \in \mathtt{Rel}(S_k, k)} \{(x,\ relay(x,\ j)),\ (relay(x,\ j),\ x)\}$. Note that $\mathbf{H}(S_k,\ k)$ is a simple directed graph, i.e., there are no multiple edges between two vertices.

(2) The *RPS request probability matrices* $\mathbf{R}_x$ for $\mathbf{H}(S_k,\ k) = (\mathbf{V},\ \mathbf{E})$ and $x \in S_k$, where, $\mathbf{R}_x[v_k, v_\ell]$ is the probability that the process at vertex $v_u \in \mathbf{V}_x$ forwards the client request to the server at vertex $v_\ell \in \mathbf{V}_x$, for an edge $(v_u,\ v_\ell)$ in $\mathbf{E}_x$.

(3) The *RPS blocking periods*, $(\mathbf{B}_x)_{x \in S_k}$, where $\mathbf{B}_x$ is a function over $\mathbf{V}_x$, and $x \in S_k$ refers to client or server processes. Note that the edges in $\mathbf{E}_x^i$ form a directed circle in the graph $\mathbf{H}(S_k, k)$, where $x \in S_k$ and $i \in [1, 3]$. A client request of $x \in \mathbf{V}_x$ to server $o[k]$, is blocking $o[k]$ for $\mathbf{B}_x(o[k]) = x.\mathbf{B}[k]$ time. A client request of $x \in \mathbf{V}_x$ to server $relay(x, j)$, blocks that server for a period of $\mathbf{B}_x(relay(x,\ j)) = x.\mathbf{D}[j]$, if $j \in [k+1, M]$ and $\mathbf{B}_x(relay(x,\ j)) = \mathbf{W}(x,\ relay(x, j),\ o[k])$ otherwise $(j \in \mathtt{Rel}(S_k, k) \setminus [k+1, M])$. Once the servers return to process $x \in \mathbf{V}_x$ with an answer, after a period of $\mathbf{B}_x(x) = \mathbf{T}_x$, process x sends a new client request to a server in $\mathbf{V}_x \setminus \{x\}$. Note that $x.\mathbf{B}[k]$, $x.\mathbf{D}[j]$, $\mathbf{W}(x,\ relay(x, j),\ o[k])$ and $\mathbf{T}_x$ are functions of x and k, x and j, x, $relay(x, j)$ and $o[k]$, and respectively, x.

Note that this definition of an RPS is a special case of the definition given in [18], which is adapted to our purposes. A contention subsystem $\mathcal{CS}(S_k, k)$ is directly mapped to an RPS $\mathcal{RP}(\mathbf{S}_k, k)$, by setting $o[k] = object[k]$, $t[n] = thread[n]$, a message to be a demand request, $\mathbf{H}(S_k,$

$k) = \mathcal{H}(S_k, k)$, $(\mathtt{R}_x)_{x \in \mathtt{S}_k} = (\mathcal{R}_s)_{s \in S_k}$ and $(\mathtt{B}_x)_{x \in \mathtt{S}_k} = (\mathcal{B}_s)_{s \in S_k}$, where $k \in [1, M]$ and $n \in [1, N]$. Namely, we set $x.\mathtt{B}[k] = s.B[k]$, $x.\mathtt{D}[j] = s.D[j]$, $\mathtt{W}(x, relay(x, j), o[k]) = W(x, relay(s, j), object[k])$ and $\mathtt{T}_x = \mathcal{T}_s$, where $x \in \mathtt{S}_k$ equals to the respective $s \in S_k$ following the mapping that we described above.

Lemma 9. *The running time of each framework iteration is* $O(M \cdot N^4)$.

Proof. The running time per iteration of the Baynat and Dallery algorithm [4] is in $O(|stations| \cdot |classes|^3)$, where $|stations|$ and $|classes|$ are the numbers of the network's stations and classes, respectively. In the context of shared-object systems, the number of *stations*, $|stations|$, corresponds to the number of vertices $|\mathcal{V}|$ of the contention graph $\mathcal{H}(S_k, k) = (\mathcal{V}, \mathcal{E})$ and the number of classes to S_k's cardinality, where $k \in [1, M]$. Thus, the running time of each iteration is in $O(|\mathcal{V}| \cdot |S_k|^3)$. Notice that $|\mathcal{V}| = N + N \cdot (M - 1) + 1$ and $|S_k| = N$, if $S_k = thread$, and $|\mathcal{V}| = M - \ell + 1$ and $|S_k| = 1$, if $S_k = \{object[\ell]\}$, where $\ell \in [1, k - 1]$. The result follows by taking the maximum $|\mathcal{V}|$ and $|S_k|$ of these two cases. ∎

Corollary 10. *Lemmata 7, 8 and 9 imply Theorem 6.*

9   Finding an ε-OSE

We present a procedure (Algorithm 3) for finding ε-OSEs. We give a detailed explanation of Algorithm 3 (Section 9.1) and analyze its running time (Lemma 11 of Section 9.5). We also detail the algorithm's functions (sections 9.2, 9.3, and respectively, 9.4), which are $initializeSystemState()$, $updateStates()$, and $recalcB()$ and $recalc\mathcal{T}()$.

9.1   The ε-OSE solver

The procedure always halts and computes an approximated equilibrium, ε-OSE, when such is reachable. Namely, whenever the job arrival and completion rates become equal, the procedure returns the system state in an ε-OSE, or indicates that an OSE is not a state that the system can be in. The procedure sets initial values to the system state, $c[s, d]$, and then uses the proposed methods (sections 5 to 8) for estimating $c[s, d]$ iteratively, until convergence. The decision on when to stop considers the system inter-demand period, $\{\mathcal{T}_{item}\}_{item \in V \setminus \{object[M]\}}$, and stops whenever there is no $item \in V \setminus \{object[M]\}$ for which the change in $\mathcal{T}_{item}$ is greater than ε since the previous iteration, where $\mathscr{G} = (V, E)$ in the acquisition graph.

The procedure's input includes the system parameters, i.e., number of threads N and objects M, the jobs, job_i, $i \in [1, J]$, and their arrival rates $\{\lambda_{i,n}\}_{i \in [1,J], n \in [1,N]}$ to each $thread[n]$, $n \in [1, N]$, as well as the request probability matrix, R. The procedure's output includes the delay D, inter-demand period T and blocking period B between all system items, as well as, the item inter-demand periods $\mathcal{T}_v$, for every item $v \neq object[M]$.

The procedure starts with a system state that represents the case in which all queues are empty (see the function $initializeSystemState()$). It then estimates the state of a system in which threads can block one another, and the delay grows as more requests are pending in the queues. The main part of the pseudocode (Algorithm 3) consists of a repeat-until loop (lines 8–13) that follows the procedure's initialization (line 6). Before the procedure can return its output value, the loop has to end either when the ε-OSE conditions are satisfied or when the procedure detects that an OSE

cannot be reached. Each iteration aims at further improving the ε-OSE estimation. The repeat-until loop then exits when no item changes by at least ε between every two iterations (and thus the system state satisfies the conditions of an approximated equilibria).

In every iteration, the procedure computes (i) the blocking periods of demand requests for objects, $s.B[k]$ (the function $updateStates()$), (ii) the thread inter-demand periods, $\mathcal{T}_{thread[n]}$ (the function $augmntThreadBlock()$), and (iii) the object inter-demands, $\mathcal{T}_{object[j]}$ (which is the function $updateStates()$), where $(s, object[k])$ is an edge of the acquisition graph $\mathscr{G}$, $n \in [1, N]$ and $j \in [1, M-1]$. The repeat-until loop repeats the steps (i), (ii) and (iii), which deals with inter-dependencies using alternating backward and forward iterations. Namely, it resolves the forward dependencies in which $(s, object[k])$'s blocking period, $s.B[k]$, depends on $object[\ell]$'s delay by iterating backward, where $\ell \in (k, M]$, i.e., starting from $k = M$ and counting downwards, we can estimate $s.B[k]$, because (in a system that its state satisfies the equilibrium conditions) all of $(s, object[k])$'s forward dependencies can be resolved. Similarly, its uses forward iterations for resolving backward dependencies with respect to d's item inter-demand period, $\mathcal{T}_{object[k]}$, because all of $object[k]$'s backward dependencies are resolved. Moreover, the function $augmntThreadBlock()$ allow the repeat-until loop to stop whenever the job inter-arrival time becomes less or equal than the time it takes that thread to complete such jobs, i.e., there's no OSE.

The blocking period estimation, step (i), starts from the last system object, $object[M]$, where there are no dependencies on the delay of subsequent demand requests. For every s such that $(s, object[M])$ is an edge of $\mathscr{G}$, we calculate the delay, $s.D[M]$, and the pairwise inter-demand period, $s.T[M]$, of demand requests to $object[M]$ through the $updateStates(\#B, k, thread, object, \{\mathcal{T}_v\}_{v \in allStates}, R)$ function (line 9), where $allStates = \{thread[n] \mid n \in [1, N]\} \cup \{object[k] \mid k \in [1, M-1]\}$. Therefore, the procedure can compute the blocking period of demand requests to $object[M-1]$, $s.B[M-1]$, because it has just estimated the dependencies of subsequent demand requests for $object[M]$. Repeating this process for $k = M-1, \ldots, 1$, the procedure compute the blocking periods of demand requests to any $object[k]$, $s.B[k]$, where $k \in [1, M]$. Note that this is possible, since in every step k, $k = M-1, \ldots, 1$, of this for loop, we have already computed the demand request delays and the inter-demand periods of every $object[k']$, $object[k].D[k']$, and respectively, $object[k].T[k']$, where $k' \in [k+1, M]$ (Section 6).

After computing the blocking periods, delays and pairwise inter-demand periods, the procedure estimates the threads' inter-demand periods (Section 7), which can be used to estimate the job completion rates. It does this in step (ii), through the function $augmntThreadBlock(\Sigma_i \lambda_{i,n}, blocking(n))$ (line 10) and by using the thread idle probabilities (Appendix 4.1), where $blocking(n) = A + \Sigma_{k=1}^M R(thread[n], object[k]) \cdot thread[n].D[k]$. Moreover, note that the $augmntThreadBlock()$ function (Section 7) allows the repeat-until loop to stop whenever it detects that the inter-arrival time of jobs to a thread becomes less or equal than the time it takes that thread to complete such jobs (i.e., there's no OSE). Note that the repeat-until loop breaks when the Boolean variable $loopEnd$ is true (line 10).

Step (iii) uses the $updateStates()$ function (line 11) for calculating the object inter-demand periods, after calculating the new estimates for the delay and the pairwise inter-demand periods. The procedure estimates the inter-demand period $\mathcal{T}_{object[k]}$, for each $object[k]$, $k \in [1, M-1]$, via the item inter-demand period of demand requests for items that precede $object[k]$ in a job path ($\bullet$, $item[j]$, $object[k]$, $\bullet$) (Section 7). Therefore, the procedure estimates the inter-demand periods for each $object[k]$ in the order of $k = 1, \ldots, M-1$.

Once the procedure verifies the satisfaction of the ε-OSE conditions (Section 2), the repeat-until loop end (line 12) and the procedure returns.

9.2 The $initializeSystemState()$ function

The procedure $initializeSystemState()$ initializes Algorithm 3 assuming that there is no contention, i.e., all queues have zero length in the shared-object system (lines 6 and 14). In the context of shared-object systems, no contention means that the delay of each demand request equals its blocking period, $s.D[k] = s.B[k]$, for all items s and $d = object[k]$. Thus, the procedure initializes the blocking periods $s.B[k]$ through the function $initRecord()$ (lines 15–16). It uses the average demand completion period $f_{s,object[k]}$, as proposed in Section 6, which is a constant. That is, $s.B[M] = A + R(s,d) \cdot R(d,d) \cdot f_{s,d}$, if $d = object[M]$ and $s.B[k] = A + R(s,d) \cdot R(d,d) \cdot f_{s,d} + \Sigma_{\ell=k+1}^{M} R(d, object[\ell]) \cdot d.D[\ell]$, if $d = object[k]$, $k \in [1, M-1]$, where $s \in \{thread[n] \mid n \in [1,N]\} \cup \{object[i] \mid i \in [1, k-1]\}$. Namely, the blocking period of a request to access (demand) the last object equals to the acquisition period plus the average demand completion period on that object and the blocking period of a demand to $d = object[k]$, $k \in [1, M-1]$, is recursively computed as the sum of the acquisition period, A, the average demand completion period $R(s,d) \cdot R(d,d) \cdot f_{s,d}$ plus the average blocking period of a subsequent demand to one of the following objects $d.B[\ell]$, $\ell \in [k+1, M]$, weighted by the probability of sending such a demand, $R(d, object[\ell])$, $\ell \in [k+1, M]$. Moreover, the $s.T[k]$ pairwise inter-demand periods are set to equal $s.B[k]$.

The $\mathcal{T}_{thread[n]}$ (thread) inter-demand periods are computed (line 17) through the function $augmentThreadBlock(\Sigma_i \lambda_{i,n}, blocking(n))$, where $blocking(n)$ is defined in line 4. Furthermore, the $\mathcal{T}_{object[j]}$ (object) inter-demand periods are set to $\Sigma_{\ell=j+1}^{M} R(object[j], object[\ell]) \cdot object[j].T[\ell]$ (line 18). Algorithm 3 iteratively finds the correct values of the pairwise and item inter-demand periods (arbitrary initialization of the system's state is proposed in [18]).

9.3 The $updateStates()$ function

This function updates the blocking periods and the item inter-demand periods with respect to $object[k]$ (lines 19–25). If the input tag is $\#B$, the function updates the blocking periods $s.B[k]$ for every $s \in \{thread[n] \mid n \in [1, N]\} \cup \{object[j] : j \in [1, k-1]\}$, as in Section 6. Otherwise, if the input tag is $\#\mathcal{T}$, the function updates the inter-demand period of $d = object[k]$, $\mathcal{T}_d$, as in Section 7.

The procedure $updateStates()$ defines the contention subsystem for every set of item sources S_k using the function $defContentionSubsystem()$ (line 20 if $S_k = thread$ and line 23 if $S_k = \{object[j]\}$, $j \in [1, k-1]$). It then calculates the delay, $s.D[k]$, and pairwise inter-demand period, $s.T[k]$, for every $s \in S_k$ using the $BDF()$ function of Appendix 4.2 (line 21 if $S_k = thread$ and line 24 if $S_k = \{object[j]\}$, $j \in [1, k-1]$). The procedure ends with the computation of the blocking periods or the inter-demand periods, depending on the tag with which the procedure was called.

9.4 The $recalcB()$ and $recalc\mathcal{T}()$ functions

We present the exact formulas that give the first three moments of $s.B[k]$, i.e., the function $recalcB()$, and $\mathcal{T}_d$, i.e., the function $recalc\mathcal{T}()$, where $d = object[k]$. We find these formulas using the equations in sections 6, and respectively, 7.

Let $F_{s,d} = A + R(s,d) \cdot R(d,d) \cdot f_{s,d}$, be the sum of the acquisition time and the job completion period times the related probabilities ($F_{s,d}$ can be bound by a constant and thus can be treated as such). The first three moments of the blocking time are $E(s.B[M]^m) = (F_{s,M})^m$, for $m = 1, 2, 3$, and $E(s.B[k]) = F_{s,d} + \Sigma_{k'=k+1}^{M} (\Pr[(s, \bullet, d, object[k'])] \cdot E(d.B[k'])$, $E(s.B[k]^2) = (F_{s,object[d]})^2 + \Sigma_{k'=k+1}^{M} \Pr[(s, \bullet, d, object[k'])] (E(d.D[k']^2) + 2F_{s,d} \cdot E(d.D[k']))$ and $E(s.B[k]^3) = (F_{s,d})^3 + \Sigma_{k'=k+1}^{M} \Pr[(s, \bullet, d, object[k'])] (E(d.D[k']^3) + 3(F_{s,d})^2 \cdot E(d.D[k']) + 3F_{s,d} \cdot E(d.D[k']^2))$, if $d =$

$object[k]$ and $k \in [1, M-1]$. Moreover, the first three moments of $\mathcal{T}_d$, where $d = object[k]$, are $E(\mathcal{T}_d)$ $= (\Sigma_{s \in arrivals(k)} \ \omega(s, d) \cdot (F_{s,d} + E(\mathcal{T}_s))) \ / \ (\Sigma_{s \in arrivals(k)} \ \omega(s, d))$, $E(\mathcal{T}_d^2) = (\Sigma_{s \in arrivals(k)} \ \omega(s,$ $d) \cdot ((F_{s,d})^2 + E(\mathcal{T}_s^2) + 2F_{s,d} \cdot E(\mathcal{T}_s)) \ / \ (\Sigma_{s \in arrivals(k)} \ \omega(s, d))$ and $E(\mathcal{T}_d^3) = (\Sigma_{s \in arrivals(k)} \ \omega(s,$ $d) \cdot ((F_{s,d})^3 + E(\mathcal{T})_s^3) + 3(F_{s,d})^2 E(\mathcal{T}_s) + 3F_{s,d} E(\mathcal{T}_s^2)) \ / \ (\Sigma_{s \in arrivals(k)} \ \omega(s, d))$. We remind that $arrivals(object[k]) = thread \cup \{object[j] \ | \ j \in [1, k-1]\}$ and $\omega(s, object[k]) = R(s, object[k]) \cdot s.T[k]$ (Section 7).

9.5 Running time

Notice that the running times of Algorithm 2 (Baynat-Dallery framework) and Algorithm 3 depend on the number of iterations of these algorithms. Lemma 11 bounds the procedure running time for one iteration.

Lemma 11. *The running time of one iteration of Algorithm 3 is in* $O(M^2 \cdot N^4 + M^3)$.

Proof. We look at the running time of each step of the repeat-until loop to find the algorithm's running time. Steps (i) and (iii) call the function $updateStates()$ so at most M; M, and respectively $M - 1$ times. Note that the function $updateStates()$ calls at most $1 + (M - 1)$ times the function $BDF()$, because the input parameter k, which denotes the object whose state is to be updated, is at most M (line 22). The first call is done by setting $S_k = thread$ and (at most) $M - 1$ calls are done by setting $S_k = \{object[j]\}$, $j \in [1, k - 1]$. The $BDF()$ function has running time in $O(M \cdot N^4 \cdot I_f)$ and $O(M \cdot I_f)$ (Lemma 9), when $BDF()$ is called for $S_k = thread$, and respectively, for $S_k = \{object[j]\}$, $j \in [1, k - 1]$, where I_f denotes the maximum number of framework iterations (Lemma 9). It also holds that the function $augmntThreadBlock()$ is called N times in step (ii) and its the running time is practically constant (see Appendix 4.1 with respect to the findings of Latouche and Ramaswami [13]). Thus, the running time of one iteration of Algorithm 3 is in $O(M \cdot (M \cdot N^4 \cdot I_f) + N + M \cdot ((M - 1) \cdot M \cdot I_f)) = O(M^2 \cdot N^4 + M^3)$.

10 Conclusions

We consider a resource allocation problems that can be modeled as generalized dynamic dining philosophers problems. We formulate questions that are associated with equilibrium situations in such systems, where input and output rates match. We believe that the way we find the equilibrium as well as estimate the delay and throughput in such systems can be the basis for an analysis of further generalizations of the problem studied here, such as the ones that are described in the literature on resource allocation, e.g., non-sequential scheduling, such as parallel resource acquisition (2-phase locking) and resource acquisition that is reactive to contention conditions [12, 15, 10, 11, 17].

Algorithm 3: The procedure for finding an ε-OSE

---

1 Input: Number of objects, M and threads, N; jobs $\{job_i\}_{i\in[1,J]}$ and their arrival rates to threads $\{\lambda_{i,n}\}_{n\in[1,N],i\in[1,J]}$; R request probability matrix;

2 Variables: The item states are recorded in the arrays $thread[1,N]$ and $object[1,M-1]$, such that $thread[n]$ is of the form $\langle$(inter-demand period) $T[1,M]$, (delay) $D[1,M]$, (blocking period) $B[1,M]\rangle$ and $object[k]$ is of the form $\langle$(inter-demand period) $T[k+1,M]$, (delay) $D[k+1,M]$, (blocking period) $B[k+1,M]\rangle$; $\mathcal{T}_v$, item v's inter-demand period, for every $v = thread[n]$ such that $n \in [1,N]$ and $v = object[k]$ such that $k \in [1,M-1]$; $loopEnd$ (Boolean) this variable is true when the function $augmentThreadBlock()$ decides that no OSE can be found and thus the loop should stop.

3 Output: $(thread[1,N], object[1,M-1])$

4 Macros: $objThrdSet() = \{\langle tag, i, \tau\rangle | (tag = \#o \wedge \mathcal{T}_{object[i]} = \tau) \vee (tag = \#t \wedge \mathcal{T}_{thread[i]} = \tau)\}$
$converged(prev, curr) = (\nexists \langle tag, i, \tau\rangle \in prev, \langle tag, i, \tau'\rangle \in curr : |\tau - \tau'| \geq \varepsilon)$
$blocking(n) = A + \Sigma_{k=1}^{M} R(thread[n], object[k]) \cdot thread[n].D[k]$
$allStates = \{thread[n] | n \in [1,N]\} \cup \{object[k] | k \in [1,M-1]\}$

5 begin

6 $\quad$ $initializeSystemState(N, M, \{job_i\}_{i\in[1,J]}, \{\lambda_{i,n}\})$

7 $\quad$ $loopEnd \leftarrow$ **false**

8 $\quad$ **repeat**

9 $\quad\quad$ **let** $prevSet \leftarrow objThrdSet()$ **for** $k = M$ **to** 1 **do**
$\quad\quad$ $updateStates(\#B, k, thread, object, \{\mathcal{T}_v\}_{v\in allStates}, R)$;

10 $\quad\quad$ **foreach** $n \in [1,N]$ **do**
$\quad\quad$ $(\mathcal{T}_{thread[n]}, loopEnd) \leftarrow augmntThreadBlock(\Sigma_i \lambda_{i,n}, blocking(n))$;

11 $\quad\quad$ **for** $k = 1$ **to** $M - 1$ **do** $updateStates(\#\mathcal{T}, k, thread, object, \{\mathcal{T}_v\}_{v\in allStates}, R)$;

12 $\quad$ **until** $converged(prevSet, objThrdSet()) \vee loopEnd =$ **true**;

13 $\quad$ **return** $(thread[1,N], object[1,M-1])$

14 procedure $initializeSystemState(N, M, \{job_i\}_{i\in[1,J]}, \{\lambda_{i,n}\})$ **begin**

15 $\quad$ **for** $k = M$ **to** 1 **do**

16 $\quad\quad$ **foreach** *item s such that $(s, object[k])$ is an edge in $\mathscr{G}$* **do**
$\quad\quad$ $s \leftarrow initRecord(s, k, thread[1,N], object[1,M-1])$;

17 $\quad$ **for** $n = 1$ **to** N **do** $\mathcal{T}_{thread[n]} \leftarrow augmentThreadBlock(\Sigma_i \lambda_{i,n}, blocking(n))$;

18 $\quad$ **for** $k = 1$ **to** $M - 1$ **do** $\mathcal{T}_{object[k]} \leftarrow \Sigma_{\ell=k+1}^{M} R(object[k], object[\ell]) \cdot object[k].T[\ell]$;

19 procedure $updateStates(tag, k, thrSet, objSet, (\mathcal{T}_v)_{v\in allStates}, R)$ **begin**

20 $\quad$ $\mathcal{CS}(thrSet, k) \leftarrow$
$\quad$ $defContentionSubsystem(thread[1,N], object[1,M-1], k, (\mathcal{T}_v)_{v\in allStates}, R)$;

21 $\quad$ $(thrSet[n].T[k], thrSet[n].D[k])_{n\in[1,N]} \leftarrow BDF(\mathcal{CS}(thrSet, k))$;

22 $\quad$ **for** $j = 1$ **to** $k - 1$ **do**

23 $\quad\quad$ $\mathcal{CS}(\{objSet[j]\}, k) \leftarrow$
$\quad\quad$ $defContentionSubsystem(\{objSet[j]\}, object[1,M-1], k, (\mathcal{T}_v)_{v\in allStates}, R)$

24 $\quad\quad$ $(objSet[j].T[k], objSet[j][n].D[k]) \leftarrow BDF(\mathcal{CS}(thrSet, k))$

25 $\quad$ **if** $tag = \#B$ **then** $recalcB(k)$, **else if** $tag = \#\mathcal{T}$ **then** $recalc\mathcal{T}(k)$;

References

[1] I. Adan and J. Resing. *Queueing theory*. Eindhoven University of Technology Eindhoven, 2002. Can be accessed via `www.win.tue.nl/~iadan/queueing.pdf`.

[2] T. Altiok. *Performance analysis of manufacturing systems*. Springer, 1997.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM (JACM)*, 22(2):248–260, 1975.

[4] B. Baynat and Y. Dallery. A product-form approximation method for general closed queueing networks with several classes of customers. *Perform. Eval.*, 24(3):165–188, 1996.

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[6] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Communications of the ACM*, 16(9):527–531, 1973.

[7] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *SIGMOD Conference*, pages 117–128. ACM, 2000. SIGMOD Record 29(2), June 2000.

[8] R. M. Feldman and C. Valdez-Flores. *Applied probability and stochastic processes*. Springer, 2010.

[9] W. J. Gordon and G. F. Newell. Closed queuing systems with exponential servers. *Operations research*, 15(2):254–265, 1967.

[10] P. H. Ha, M. Papatriantafilou, and P. Tsigas. Self-tuning reactive distributed trees for counting and balancing. In T. Higashino, editor, *Principles of Distributed Systems, 8th International Conference, OPODIS 2004, Grenoble, France, December 15-17, 2004, Revised Selected Papers*, volume 3544 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2004.

[11] P. H. Ha and P. Tsigas. Reactive multi-word synchronization for multiprocessors. *Journal of Instruction-Level Parallelism*, 6:1–25, 2004.

[12] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier, 2012.

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

[14] N. A. Lynch. Upper bounds for static resource allocation in a distributed system. *J. Comput. Syst. Sci.*, 23(2):254–278, 1981.

[15] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[16] M. F. Neuts. *Matrix-geometric solutions in stochastic models - an algorithmic approach*. Dover Publications, 1994.

[17] M. Papatriantafilou and P. Tsigas. On distributed resource handling: Dining, drinking and mobile philosophers. In A. Bui, M. Bui, and V. Villain, editors, *On Principles Of Distributed Systems, Proceedings of the 1997 International Conference, Chantilly, France, December 10-12, 1997*, pages 293–308. Hermes, 1997.

[18] S. Ramesh and H. G. Perros. A multi-layer client-server queueing network model with non-hierarchical synchronous and asynchronous messages. *Perform. Eval.*, 45(4):223–256, 2001.