Approximating Fork-Join Systems via Mixed Model Transformations

Rares-Andrei Dobre
Department of Computing
Imperial College London
London, UK
rares.dobre22@imperial.ac.uk

Zifeng Niu
Department of Computing
Imperial College London
London, UK
zifeng.niu19@imperial.ac.uk

Giuliano Casale
Department of Computing
Imperial College London
London, UK
g.casale@imperial.ac.uk

ABSTRACT

While product-form queueing networks are effective in analyzing system performance, they encounter difficulties in scenarios involving internal concurrency. Moreover, the complexity introduced by synchronization delays challenges the accuracy of analytic methods. This paper proposes a novel approximation technique for closed fork-join systems, called MMT, which relies on transformation into a mixed queueing network model for computational analysis. The approach substitutes fork and join with a probabilistic router and a delay station, introducing auxiliary open job classes to capture the influence of parallel computation and synchronization delay on the performance of original job classes. Evaluation experiments show the higher accuracy of the proposed method in forecasting performance metrics compared to a classic method, the Heidelberger-Trivedi transformation. This suggests that our method could serve as a promising alternative in evaluating queueing networks that contains fork-join systems.

CCS CONCEPTS

• **Software and its engineering** → **Software performance**; • **Networks** → **Network performance modeling**.

KEYWORDS

Queueing Network, Fork-Join system, Synchronization Delay

ACM Reference Format:
Rares-Andrei Dobre, Zifeng Niu, and Giuliano Casale. 2024. Approximating Fork-Join Systems via Mixed Model Transformations. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652277

1 INTRODUCTION

Modern software systems are typically implemented in a distributed manner to improve performance, reliability, and scalability [18]. Under this pattern, parallel and concurrent structures have gained increased importance over the years [9]. The software components are often parallelized to achieve higher execution efficiency and increase the utilization of the available resources. Parallel jobs typically follow a fork-join mechanism. A parallel job is forked into several tasks, which are executed concurrently on distinct resources within the system. After finishing the execution, a task has to await its sibling tasks at the join point. This job exits the join point and continues only when all its tasks have finished execution.

Queueing networks are a class of efficient performance models to understand the impact of execution mechanisms on system performance. A large number of computer systems can be abstracted as product-form queueing networks from which designers obtain accurate performance predictions [3]. Nevertheless, product-form queueing network models do not accommodate jobs that feature internal concurrency. Furthermore, the exact analysis for internal concurrency within a queueing network can rapidly lead to a state-space explosion.

In addition to the time spent on service, the total time of a parallel execution involves two other delays: queueing delay and synchronization delay [19]. Synchronization delay occurs on any completed task that waits for the completion of other sibling tasks before leaving the fork-join system. This coordination introduces dependencies and leads to an increased complexity in designing an accurate analysis for parallel executions. Therefore, it is necessary to have approximate analytic performance models to analyze parallelism for software designers and practitioners.

The technique developed by Heidelberger and Trivedi [13] is long established. This method, which we refer to as HT method in the rest of the paper, decomposes a job into a primary task and a fixed number of secondary tasks; these tasks are assumed to be independent of each other and belong to different job classes. Then probabilistic routing and pseudo-servers are used to replace parallel executions so that a product-form solution can be produced.

In this paper, we propose a novel approach, named MMT, for the analysis of fork-join systems. Similarly to the HT method, the fork and join are replaced by a probabilistic router and a delay, respectively. In our terminology, a probabilistic router is an abstract node that routes incoming jobs along output branches, according to a probabilistic routing policy and with zero service time. Beyond that, it introduces auxiliary open job classes to mimic the influence of parallel computation and synchronization delay on original job classes, which leads to a mixed queueing network model to analyze. The arrival rates and service rates of the auxiliary classes are computed from an iterative procedure that enables us to finally obtain the approximated solution of the original queueing network containing parallelism. The main performance measures in the fork-join system can be obtained using the method described.

The effectiveness of the proposed method is validated by comparison with simulations and the HT method. We use the simulation results as ground truth. Experiments are conducted on closed queueing networks with homogeneous or heterogeneous fork-join queues

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652277

Table 1: Notations for considered queueing networks

Symbol	Definition
r	index of the job class
m	index of the service station
c	index of the concurrent (fork-join) structure
$\mu_{m,r}$	class-r service rate at station m
λ_r	class-r arrive rate (for open class)
$Q_{m,r}$	class-r queue length at station m
$X_{m,r}$	class-r throughput at station m
$T_{m,r}$	class-r response time at station m
$U_{m,r}$	class-r utilization at station m
K	number of job classes in the network
M	number of service stations in the network
C	number of fork-join structures in the network
F_c	number of parallel paths spawned by the fork of the structure c

(i.e., where servers can have the same or different service rates). Compared to the HT method, the proposed method offers lower error rates on predicted performance measures.

The rest of the paper is organized as follows. In Section 2, we provide background on queueing network theory, fork-join systems, and their analysis. In Section 3, we propose our MMT method for the analytic analysis of fork-join systems. In Section 4, we present evaluation experiments and results. In Section 5, we review related work. Finally, we conclude the paper in Section 6.

2 BACKGROUND

2.1 Queueing Networks

Queueing networks serve as a class of models for analyzing the performance of systems. They are made up of a collection of service stations indexed by $m = 1, \ldots, M$, in which jobs are queued and executed. A service station could have either infinite servers or finite servers, and is referred to as delay or queue station, respectively. The common use of delay stations is to represent the think times of workloads, while queue stations typically represent system resources [16]. In a queue station, there may be competition among jobs for the server, resulting in waiting times. A queueing network may execute multiple classes of jobs indexed by $r = 1, \ldots, K$. Distinct job classes typically feature different service rates $\mu_{m,r}$, and can be further categorized into closed and open classes. For closed job classes, a fixed number of jobs circulate within the network. For open job classes, jobs from a source continuously arrive to the network with rate λ_r. Scheduling policies determine the order in which jobs are served, with common policies including First-Come First-Serve (FCFS) and Processor Sharing (PS).

To evaluate the performance of a queueing network, one approach is to solve a system of global balance equations to get state probabilities of the underlying Markov chain where all performance measures can be further obtained. However, this method becomes hardly practical on complex networks due to the state-space explosion. Among queueing networks, there is a special class named product-form queueing networks. They have local balance properties and their exact performance measures can be obtained without resorting to the underlying state space. Mean Value Analysis (MVA) [21] is a prevalent technique to solve product-form queueing networks. In scenarios where exact analysis becomes computationally challenging, Approximate Mean Value Analysis (AMVA) [8] provides a practical alternative, offering insight into key performance measures such as queue lengths $Q_{m,r}$, throughputs $X_{m,r}$, response times $T_{m,r}$, and utilizations $U_{m,r}$.

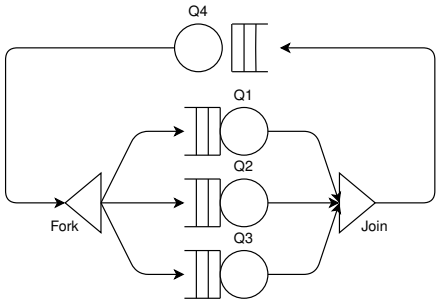


Figure 1: Queueing network containing a fork-join system

2.2 Fork-Join Systems

A queueing network may include a number of fork-join systems indexed by $c = 1, \ldots, C$. Fork-Join systems contain both fork and join nodes, which are a particular structure in queueing networks. The fork node has the property that any arriving job is split into multiple tasks to be serviced independently and in parallel, while the join node combines these tasks back into the original job after they have finished processing [23]. Let F_c denote the number of parallel paths spawned by the fork node of the system c, the F_c executions are assumed to be independent of each other so they do not intersect before reaching the join node. The notations for networks considered by this paper are summarized in Table 1.

An example of a closed queueing network containing a fork-join system is given in Figure 1. Jobs circulate between the fork-join system and a single queueing station. Any job that arrives at the fork node is split into three tasks to be processed on Q1, Q2, and Q3. After completing processing, they must wait for the sibling tasks to finish. Once all the tasks spawned by the job are finished, they are joined together back into the original job, which subsequently moves on to Q4.

2.3 Heidelberger-Trivedi Approximation Method

A queueing network that contains fork-join systems has no product-form solution [3], thus it cannot be efficiently solved in general. A classic transformation method is developed by [13], which originally transforms single-class closed queueing networks including a single parallel construct into a queueing network with no concurrency so that analytic methods such as AMVA can be applied. The main feature of this transformation is that each closed class is coupled with one auxiliary closed class for each parallel path.

The parallel constructs targeted by this work are analogous to fork-join systems. In its model, each job is represented by a primary task with multiple secondary tasks. Primary and secondary tasks are

used to represent the activity of the job executed outside and inside the fork-join system, respectively. Each primary task arriving at the fork node is forked into multiple secondary tasks, while secondary tasks arriving at the join node need to wait for their siblings to arrive before they can be joined back into the primary task.

This work associates the join node of the fork-join construct with a delay station. The time a primary task is supposed to spend at this delay station represents the overall response time of the job in the fork-join system, whereas for a secondary task it represents the corresponding synchronization delay. This approach also adds an auxiliary delay station to the network, which models the time a primary task spends outside the fork-join system. The secondary tasks use the auxiliary delay as their reference station. Their service rates are computed using an iterative procedure analogous to [14].

Suppose D_0 represents the response time of the fork-join system, R_i denotes the response time at the i-th parallel path, and F denotes the number of parallel paths spawned by fork f, the expectation of the job response time in the fork-join system is

$$E[D_0] = E[\max(R_1, \ldots, R_F)] \tag{1}$$

Therefore, the service time of the primary task at the delay station that replace the join node is $E[D_0]$. The R_i are assumed to be exponential random variables with $E[R_i] = 1/\mu_i$ for all $i \in \{1, \ldots, F\}$, thus $E[D_0]$ can be further expressed as the following [24]

$$\begin{aligned} E[D_0] = &\sum_{i=1}^{F} \frac{1}{\mu_i} - \sum_{i<j} \frac{1}{\mu_i + \mu_j} + \sum_{i<j<k} \frac{1}{\mu_i + \mu_j + \mu_k} \\ &- \cdots + (-1)^{F-1} \sum_{i_1<\cdots<i_F} \frac{1}{\mu_{i_1} + \cdots + \mu_{i_F}} \end{aligned} \tag{2}$$

where $i, j, k, i_1, \ldots, i_F$ represent path indices.

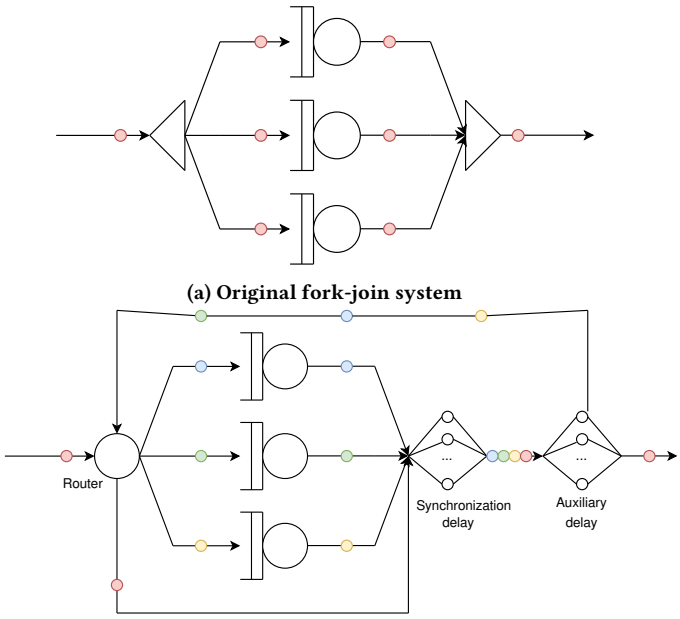
Let D_i represent the service times of the secondary tasks at this delay station, i.e. the synchronization delays, $E[D_i]$ is obtained by subtracting the mean response time of the i-th parallel path from the fork-join response time

$$E[D_i] = E[D_0] - E[R_i] \tag{3}$$

Figure 2 depicts an example of a network alteration using the aforementioned procedure. The original job class and its primary tasks are depicted with red circles in the original and transformed systems, whereas the distinct secondary tasks are represented through blue, green, and yellow circles. It can be observed from Figure 2b that the HT method produces four job classes for this single class fork-join system with three parallel paths. In the transformed fork-join system, the primary tasks are routed straight to the synchronization delay station, whereas the secondary tasks are routed through the fork-join system. From the synchronization delay station, every job is routed to the auxiliary delay station. From the auxiliary delay station, the jobs belonging to the primary task continue proceeding through the rest of the network, while the jobs from the secondary tasks are sent out to the router. The obtained model is a product-form queueing network that can be solved easily.

3 PROPOSED METHODOLOGY

The proposed method is a simpler transformation of fork-join systems by introducing only one more delay station and fewer job



(a) Original fork-join system

(b) Transforming original system into a network without concurrency

Figure 2: HT method for fork-join transformation

Table 2: Notations for auxiliary classes and network elements created by the MMT method

Symbol	Definition
r	index of the original class
r'_c	auxiliary class of r created for the fork-join structure c
o_c	router created to replace the fork node of the fork-join structure c
d_c	delay station created to replace the join node of the fork-join structure c

classes. This transformation leads to a simple yet effective computation procedure for performance measures of the original system. The new notations introduced by the proposed method are summarized in Table 2.

3.1 Network Transformation

3.1.1 Mixed model construction. We transform the network into the one that can be applied to analytic approximations. The join node of the fork-join system c is replaced by a delay station d_c, while the fork node is replaced by a router that forwards incoming jobs to original F_c parallel paths. Each path carries an equal probability $\frac{1}{F_c}$ of being selected to forward a job to, and the sum of probabilities is 1. However, a router does not offer the functionality required to spawn other tasks. Thus, even if a job is routed to a parallel path, no sibling tasks are executed concurrently on the other parallel paths. To counter this issue, the parallelism induced by a fork-join system is simulated through the addition of open job classes, which we shall refer to as the auxiliary classes. The auxiliary classes

mirror the behavior of their original classes in terms of routing and service rates within the fork-join system. As shown in Figure 3a, the auxiliary classes are routed directly to the router from the source, and they are forwarded to the sink after leaving the delay station. Thus, their impact is limited to their corresponding fork-join systems.

In contrast to the HT method, each original class passing through the fork node is attributed only one auxiliary class whose jobs are meant to act as siblings of the original job class. Due to the approximation decision of assigning path selection probabilities equal to $\frac{1}{F_c}$, the transformation exercises all parallel paths equally. This is a key difference compared to HT, as it results in a model in which classes do not map in a one-to-one fashion with a particular parallel path. Hence, any approximation error that affects an auxiliary class is equally distributed across the paths. A comparison between both HT and MMT transformations is shown in Table 3, where the original queueing network has one fork-join system and F represents the number of parallel paths.

3.1.2 Arrival rate of the auxiliary open class. Since we have introduced auxiliary open classes into the network, it is necessary to determine their arrival rates. In equilibrium, the router that replaces the fork node of the structure c satisfies the flow balance condition [16]. Let $X_{o_c,r}$ denote the class-r arrival rate/throughput at the router and assume $X_{o_c,r}$ to be the same as the class-r arrival rate at the original fork node, the class-r throughput at the original fork node is then $X_{o_c,r} \cdot F_c$. Therefore, the arrival rate of the auxiliary open class can be computed by balancing the flow at the router as

$$\lambda_{r'_c} = X_{o_c,r} \cdot (F_c - 1) \tag{4}$$

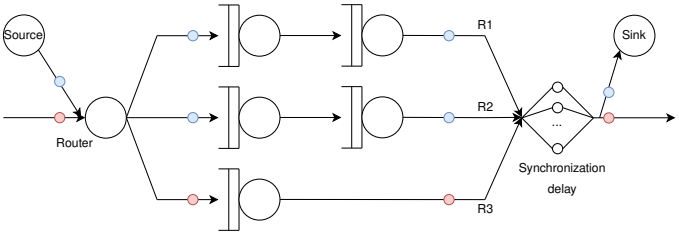
where r denotes the original job class, r'_c denotes the auxiliary class of r created for the fork-join structure c, $\lambda_{r'_c}$ represents the arrival rate of the auxiliary open class, and F_c here denotes the number of branches connected to the router.

3.2 Synchronization Delay

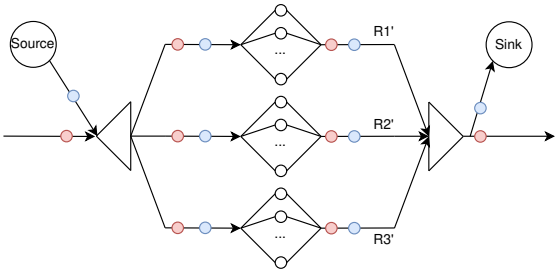
In this section, we illustrate how MMT method approximates the synchronization delay. For ease of presentation, we assume there is one fork-join system ($C = 1$) in the original network so we temporarily remove the subscript c. We assume that the response time at any parallel path p to be approximately exponentially distributed, which is proved to an effective assumption for analytic analysis [22]. Besides, let random variables $R_{1,r}, \ldots, R_{F,r}$ represent class-r response times at parallel paths $1, \ldots, F$, we assume they are mutually independent. Based on the same two assumptions as the HT method, we propose the following approximation approach consisting of two main steps to derive the synchronization delays for the fork-join system in a queueing network.

3.2.1 Mean response time at a parallel path. The first step of our approach is to obtain the response times at parallel paths of a fork-join system. For each path $p = 1, \ldots, F$, the mean response time is sum of mean response times at all queueing stations on that path.

We merge the performance measure of the auxiliary class with that of its corresponding original class. Given the merged queue length and throughput, Little's law [17] is then used to compute



(a) The network gained from the transformation of a fork-join system



(b) Corresponding homogeneous fork-join system of the netwwork in Figure 3a

Figure 3: The proposed method for fork-join transformation

the mean response time at each single station

$$R_{p,r} = \sum_{m \in M_p} \frac{Q_{m,r} + Q_{m,r'}}{X_{m,r} + X_{m,r'}} \tag{5}$$

where M_p is the set of service stations on path p.

3.2.2 Approximation by a homogeneous fork-join system. The second step is to update the service rates of the original and auxiliary classes at the sychronization delay station.

We propose to approximate the given fork-join system with a homogeneous fork-join system. As shown in Figure 3b, the corresponding homogeneous fork-join system has the same number of parallel paths and features one delay station per path. The service times of both job classes at any delay station of the new fork-join systems are assumed to be exponential random variables with the means equal to the average of the original response times of the path executions

$$E[R'_{p,r}] = \frac{E[R_{1,r}] + \cdots + E[R_{F,r}]}{F}, \quad \forall p \in \{1, \ldots, F\} \tag{6}$$

where the random variable $R'_{p,r}$ represents the class-r service time at the delay station of the parallel path p. The homogeneuous system can approximate the behavior of the orginal system since their response times are close to each other, i.e., $E[\max(R_{1,r}, \ldots, R_{F,r})] \approx E[\max(R'_{1,r}, \ldots, R'_{F,r})]$.

We therefore use $E[R'_{p,r}]$ to approximate the mean response time of each path in the original fork-join system, and then compute the synchronization delay by

$$E[D_{p,r}] = E[\max(R_{1,r}, \ldots, R_{F,r})] - E[R'_{p,r}], \quad \forall p \in \{1, \ldots, F\} \tag{7}$$

where $E[\max(R_{1,r}, \ldots, R_{F,r})]$ can be obtained by (2), and the random variable $D_{p,r}$ represents the class-r synchronization delay at the parallel path p of the original fork-join system.

Table 3: Differences between two transformation methods

	HT	MMT
added elements	2 delays	1 delay, 1 source
network type	closed	mixed
number of classes	$K(1+F)$	$2K$
synchronization delay	heterogeneous	homogeneous (after aggregation)

Algorithm 1 Procedures to solve original queueing networks

Input: a queueing network, tol
Output: $Q_{m,r}, X_{m,r}, T_{m,r}, U_{m,r}$

1: initialize $X_{o,r} \leftarrow 0$, $Q_{m,r} \leftarrow 1$, $Q^0_{m,r} \leftarrow 1$
2: $stop \leftarrow false$
3: obtain $\boldsymbol{Q}$ by fork-join transformation
4: **while** $stop$ == false **do**
5: **if** $\max\left(1\text{-}(Q^0_{m,r}/\, Q_{m,r})\right) < tol$ **then**
6: $stop \leftarrow true$
7: **else**
8: $Q^0_{m,r} \leftarrow Q_{m,r}$
9: **end if**
10: $Q_{m,r}, Q_{m,r'}, X_{m,r}\, X_{m,r'}, T_{m,r}, T_{m,r'}, U_{m,r}, U_{m,r'} \leftarrow \text{AMVA}(\boldsymbol{Q})$
11: **for** $r = 1, \ldots, K$ **do**
12: $X_{o,r} \leftarrow \text{Avg}(X_{o,r}, X_{d,r})$
13: $\lambda_{r'} \leftarrow X_{o,r} \cdot (F-1)$
14: obtain $R_{1,r}, \ldots, R_{F,r}$ by (5)
15: obtain $\mathbb{E}[D_0]$ by (2)
16: update $\mu_{d,r}$ and $\mu_{d,r'}$ by (7)
17: $Q_{m,r} \leftarrow Q_{m,r} + Q_{m,r'}$; $X_{m,r} \leftarrow X_{m,r} + X_{m,r'}$; $T_{m,r} \leftarrow Q_{m,r}/X_{m,r}$; $U_{m,r} \leftarrow U_{m,r} + U_{m,r'}$
18: **end for**
19: **end while**

3.3 Algorithm

In our method, the arrival rates of auxiliary classes and the service rates at the delay server are not known in advance. This implies that an iterative computation framework is needed to approximate the solution of the original network.

The framework is shown in Algorithm 1. For ease of presentation, we consider one fork-join system and temporarily remove the subscript c. The input of the algorithm includes an original queueing network, a tolerance tol that serves as the iteration stopping threshold, and a boolean value $stop$ that determines whether to stop the iteration. We set the initial value of $X_{o,r}$ to 0, and both $Q_{m,r}$, $Q^0_{m,r}$ to the same number (line 1), and transform the original queueing network into a product-form mixed queueing network by the proposed procedure (line 3). The stopping criterion is the difference between the queue lengths of two successive iterations (lines 5-9). At each iteration, the transformed queueing network is solved by AMVA (line 10). Then, for each auxiliary open class, we update its arrive rate (lines 12-13) and update the service rates of both auxiliary and corresponding original classes at the synchronization delay station (lines 14-16). In this way, the parameters of the queueing network are updated. The last step of an iteration is to merge the results for each original class (line 17).

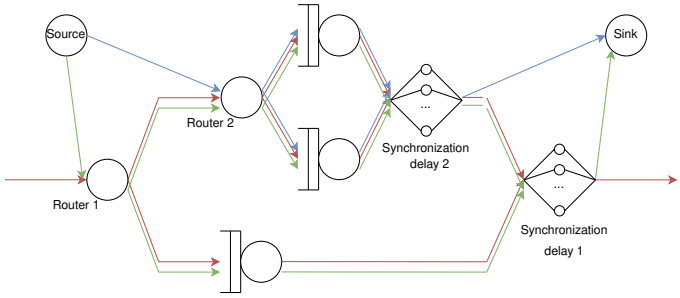


Figure 4: An example of nested fork-join transformation by the proposed method

3.4 Extend Method to Nested Fork-Join System

A nested fork-join structure is a hierarchical arrangement of fork-join systems, utilizing nested fork-join structures holds significance in the field of software development [4, 15]. Figure 4 shows a network transformed from a nested fork-join system by the proposed procedure. As it can be observed, the original system includes two fork-join structures. We refer to the outer fork-join as FJ_1 and to the inner fork-join as FJ_2. In this network, the possible paths of the original job class are depicted in red, whereas the possible paths of its auxiliary job classes are depicted in green or blue. The green paths are for the auxiliary class created for FJ_1, whereas the blue color denotes the paths of the auxiliary class created for FJ_2.

To adapt to nested fork-join systems, for each original class, we first build auxiliary classes for every fork-join structure visited by the original class. Two additions are introduced in the MMT method compared to our original method for systems without nested fork-join structures. The first addition is to start computing the synchronization delay only at the outermost fork-join structure, and then compute the nested fork-join structures recursively.

The second addition is the calculation of arrival rates of the auxiliary classes. Because nested systems incorporate inner fork-join structures and auxiliary classes are created for each of them, solely considering the throughput of the original class at an inner fork to compute the arrival rate of the auxiliary class associated to this structure is not enough. The auxiliary class created for an inner fork-join structure simulates the sibling tasks executing concurrently of the auxiliary classes created for the outer fork-join structures. In other words, the influence of both the original class and the outer auxiliary classes should be considered. Hence, before computing the arrival rate, the throughput $X_{d_c,r}$ is updated using throughputs of class-r and its auxiliary classes at that delay station

$$X_{d_c,r} = X_{d_c,r} - X_{d_c,r'_c} + \sum_{s=1}^{C} X_{d_c,r'_s} \tag{8}$$

where C denotes the set of fork-join structures in the queueing network, r'_c denotes the auxiliary class of r created for the fork-join structure c, and d_c denotes the delay station created to replace the join node of the fork-join structure c.

We refer to class-r and its auxiliary classes as *r-related classes*. Equation (8) merges the throughputs of all r-related job classes at the delay station, except X_{d_c,r'_c}, which is the throughput of the auxiliary class created for the fork-join structure associated to the

Table 4: Distinct fork-join queues used in evaluation

Groups	Topology
heterogeneous_FCFS_1	$< Q_1 \parallel Q_2 >$
heterogeneous_FCFS_2	$< Q_1 \parallel Q_2 \parallel Q_3 >$
heterogeneous_FCFS_3	$< Q_1 \parallel (Q_2 \rightarrow Q_3) >$
heterogeneous_PS_1	$< Q_1 \parallel Q_2 >$
heterogeneous_PS_2	$< Q_1 \parallel Q_2 \parallel Q_3 >$
heterogeneous_PS_3	$< Q_1 \parallel (Q_2 \rightarrow Q_3) >$
heterogeneous_PS_4	$< Q_1 \parallel Q_2 >$
heterogeneous_PS_5	$< Q_1 \parallel Q_2 \parallel Q_3 >$
homogeneous_FCFS_1	$< Q_1 \parallel Q_2 >$
homogeneous_FCFS_2	$< Q_1 \parallel Q_2 \parallel Q_3 >$
homogeneous_PS_1	$< Q_1 \parallel Q_2 >$
homogeneous_PS_2	$< Q_1 \parallel Q_2 \parallel Q_3 >$
homogeneous_PS_3	$< Q_1 \parallel Q_2 >$
homogeneous_PS_4	$< Q_1 \parallel Q_2 \parallel Q_3 >$

current delay station d_c. In other words, this equation gives the total r-related throughputs that do not leave for the sink from the current delay station. If the fork-join structure c is not nested (i.e., the outermost fork-join), X_{d_c,r'_c} will remain unchanged.

4 EVALUATION

We first compare the accuracy of the HT method [13] and MMT method against simulation results obtained by Java Modeling Tool (JMT) [2]. The implementations of these methods are included by LINE [5] that is a algorithmic framework for queueing networks and layered queueing networks [6]. The involved closed queueing networks in our evaluation can be categorised into distinct groups depending on the service rates of parallel executions (homogeneous or heterogeneous) and the scheduling used at the queueing stations (FCFS or PS).

There are two job classes in every queueing network. Both classes are closed with a population of 10 jobs each. The queueing networks always include a fork-join system containing two or three queueing stations. The service rates at these stations are randomly generated, with the average service time between 0.3 and 0.8. Table 4 shows a list of the fork-join queues used in the first experiments. We use the following notations to describe their topology: < denotes a fork node, > represents a join node, || defines a parallel branch, and → defines a serial routing.

The evaluation results are shown in Table 5. It can be observed that the proposed method achieves lower errors on most cases than the HT method. Compared to the baseline, the MMT method reduces the prediction error of queue length, response time, utilization, and throughput by 30.9%, 62.8%, 34.6%, and 35.3% on average. Figure 5a demonstrates that the two methods exhibit similar prediction accuracy on homogeneous networks, whereas Figure 5c illustrates that our method notably achieves higher accuracy on heterogeneous networks. Apart from accuracy, runtime is the other important factor to consider. As shown in Figure 5b and 5d, the average runtime of our method is less than 0.015s, substantially lower than that of the HT method, which is around 0.03s, as the runtime of the AMVA scales with the increasing number of service

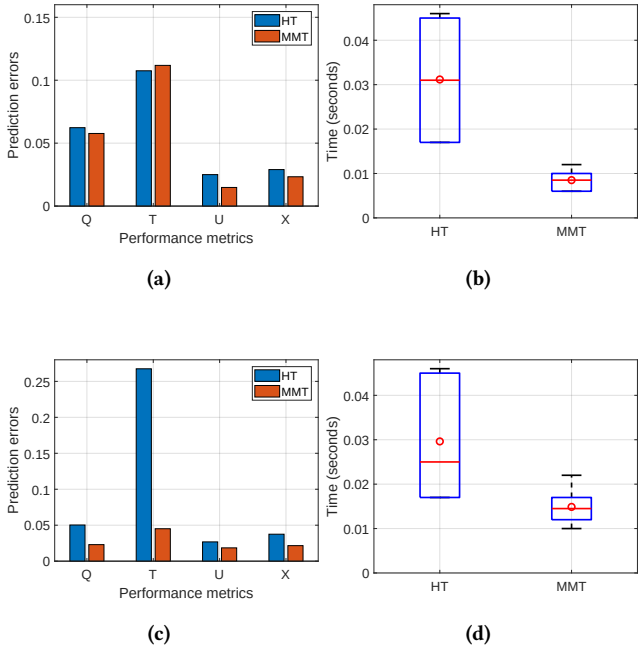


Figure 5: (a)-(b) for homogeneous networks. (c)-(d) for heterogeneous networks. (a),(c): Average prediction errors by HT and MMT methods. (b),(d): Box plots of average runtimes for both methods. Red circles and lines inside boxes represent mean and median values, respectively.

stations and job classes. Compared to our method, where only one auxiliary job class is created for a fork-join system, the HT method creates one auxiliary class for each parallel path of the concurrent system and an additional delay station to model the time spent outside the fork-join system. Hence, the models created by our method are more efficient to compute compared to those created by the HT method.

We then evaluate the MMT method on nested fork-join queues. Here we only compare the results with that of simulations since the baseline HT method is not designed for nested fork-join queues. The networks used for evaluation and numerical results are provided in Table 6 and Table 7, respectively. Figure 6 visualizes mean and maximum errors in FCFS and PS groups. As can be observed, the MMT method provides accurate predictions. This method inherently possesses the capability to handle nested fork-join systems because the auxiliary classes it creates are restricted to their corresponding fork-join systems, and they are independent of the time their original classes spend outside the fork-join systems, which is a distinct advantage of our approach. In contrast, the HT method is initially devised for a network with a single fork-join system.

5 RELATED WORK

Fork-Join queueing networks represent the key to modelling and solving parallel systems. However, they do not follow the product-form restrictions, so most algorithms devised for fork-join networks

Table 5: Queue length, response time, utilization and throughput errors of HT and MMT methods

Groups	ErrQ		ErrT		ErrU		ErrX	
	HT	MMT	HT	MMT	HT	MMT	HT	MMT
heterogeneous_FCFS_1	0.034	**0.016**	0.226	**0.022**	**0.002**	0.003	**0.004**	0.007
heterogeneous_FCFS_2	0.061	**0.029**	0.412	**0.041**	**0.004**	0.018	**0.003**	0.021
heterogeneous_FCFS_3	0.066	**0.024**	0.409	**0.034**	0.043	**0.010**	0.070	**0.012**
heterogeneous_PS_1	0.063	**0.014**	0.262	**0.031**	0.010	**0.009**	0.016	**0.010**
heterogeneous_PS_2	0.025	**0.024**	0.140	**0.070**	**0.025**	0.046	**0.027**	0.046
heterogeneous_PS_3	0.067	**0.019**	0.278	**0.046**	0.086	**0.031**	0.122	**0.039**
heterogeneous_PS_4	0.049	**0.042**	**0.070**	0.087	0.040	**0.021**	0.053	**0.022**
heterogeneous_PS_5	0.037	**0.016**	0.343	**0.030**	**0.004**	0.010	**0.005**	0.016
homogeneous_FCFS_1	**0.069**	0.078	0.125	**0.116**	0.013	**0.009**	0.021	**0.018**
homogeneous_FCFS_2	**0.083**	0.094	0.114	**0.110**	0.035	**0.018**	0.039	**0.032**
homogeneous_PS_1	0.041	**0.032**	**0.053**	0.074	0.014	**0.010**	0.021	**0.017**
homogeneous_PS_2	0.062	**0.040**	0.124	**0.116**	0.039	**0.015**	**0.025**	0.028
homogeneous_PS_3	0.053	**0.045**	**0.073**	0.099	0.025	**0.014**	0.027	**0.016**
homogeneous_PS_4	0.066	**0.057**	0.156	0.156	0.024	**0.023**	0.041	**0.029**
Mean	0.055	**0.038**	0.199	**0.074**	0.026	**0.017**	0.034	**0.022**

Table 6: Nested fork-join queues used in evaluation

Groups	Topology
FCFS_1	$< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$
FCFS_2	$< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$
FCFS_3	$< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$
FCFS_4	$< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$
PS_1	$< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$
PS_2	$< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$
PS_3	$< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$
PS_4	$< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$

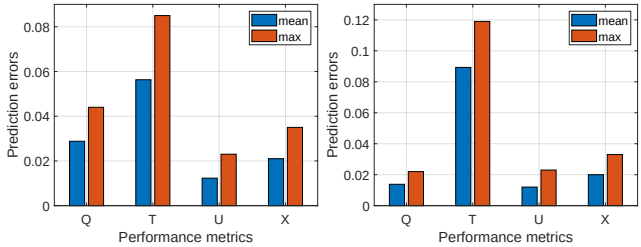


Figure 6: (a) for networks with FCFS scheduling. (b) for networks with PS scheduling.

Table 7: Queue length, response time, utilization and throughput errors of the MMT method

Groups	ErrQ	ErrT	ErrU	ErrX
FCFS_1	0.029	0.035	0.004	0.009
FCFS_2	0.023	0.036	0.004	0.007
FCFS_3	0.044	0.069	0.018	0.035
FCFS_4	0.019	0.085	0.023	0.033
PS_1	0.011	0.085	0.004	0.008
PS_2	0.012	0.070	0.004	0.007
PS_3	0.022	0.119	0.017	0.032
PS_4	0.010	0.083	0.023	0.033
Mean	0.021	0.073	0.012	0.021

are approximations and bounds, see e.g. [23],[7]. The only exact solution has been devised for two parallel servers [20]. The main difficulty in analysing fork-join queueing networks stems from the synchronization delays incurred by the jobs waiting for the other jobs created by a job to finish [11].

Duda and Czachórski [11] devise an algorithm to analyse fork-join queueing networks by replacing the fork-join constructs with load-dependent queueing stations. Its foundation consists of the flow-equivalent server method [3] and the decomposition principle. Varki [25] modifies the computation of residence time in the MVA algorithm to adapt to the closed, single-class queueing networks containing fork-join systems. This modification assumes service stations to have exponentially distributed service times and FCFS scheduling strategies. Alomari and Menasce [1] propose a method for analyzing fork-join systems involving servers with heterogeneous service times in open networks. The core of this method involves establishing bounds on the response time of a job in a fork-join system. This is achieved by analyzing the system under two scenarios: one where all stations have the same service rates and the other where their service rates vary. Mak and Lundstrom [19] introduce an iterative algorithm capable of approximating the performance measures of directed acyclic graphs abstracted from parallel systems in polynomial space and time. Franks and Woodside [12] illustrate the capability of the layered modelling framework in which a platform for defining models with parallelism can be conveniently used.

6 CONCLUSION

The paper proposes an accurate and computationally efficient approach for analyzing closed queueing networks containing fork-join systems. The core of this method involves establishing auxiliary open job classes to simulate the behavior of the original parallelism. This transformation leads to a mixed queueing network model that can be solved by analytic method. Compared to the well-established Heidelberger-Trivedi method, which uses an one-to-one fashion to create auxiliary closed job classes for each parallel path, our approach produces one auxiliary open job classes for the entire fork-join systems. The evaluation results show that our method achieves lower error rates. Meanwhile, the proposed method is faster than the baseline method since our transformed network has less number of job classes that requires less analytic computations. In addition, the design of our transformation enables us to deal with nested fork-join system, which is a notable advantage. Hence, this paper contributes a simple yet effective fork-join transformation which has the potential to be of great value in areas of the concurrent system research. An extended version of the work presented in this paper is available in [10].

REFERENCES

[1] Firas Alomari and Daniel A Menasce. 2013. Efficient response time approximations for multiclass fork and join queues in open and closed queuing networks. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2013), 1437–1446.

[2] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. 2009. JMT: performance engineering tools for system modeling. *ACM SIGMETRICS Performance Evaluation Review* 36, 4 (2009), 10–15.

[3] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. 2006. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications.* John Wiley & Sons.

[4] Michael Burke, Ron Cytron, Jeanne Ferrante, and Wilson Hsieh. 1989. Automatic generation of nested, fork-join parallelism. *The Journal of Supercomputing* 3 (1989), 71–88.

[5] Giuliano Casale. 2020. Integrated performance evaluation of extended queueing network models with line. In *2020 Winter Simulation Conference (WSC)*. IEEE, 2377–2388.

[6] Giuliano Casale, Yicheng Gao, Zifeng Niu, and Lulai Zhu. 2023. LN: A Flexible Algorithmic Framework for Layered Queueing Network Analysis. *ACM Transactions on Modeling and Computer Simulation* (2023).

[7] Giuliano Casale, Richard Muntz, and Giuseppe Serazzi. 2008. Geometric bounds: A noniterative analysis technique for closed queueing networks. *IEEE Trans. Comput.* 57, 6 (2008), 780–794.

[8] K Mani Chandy and Doug Neuse. 1982. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Commun. ACM* 25, 2 (1982), 126–134.

[9] David Culler, Jaswinder Pal Singh, and Anoop Gupta. 1999. *Parallel computer architecture: a hardware/software approach.* Gulf Professional Publishing.

[10] Rares-Andrei Dobre. 2023. *Stochastic Modelling in JLINE: Redesigning and Augmenting the MVA Solver with Fork-Join Analysis Methods.* Technical Report. MSc Final project, Department of Computing, Imperial College London.

[11] Andrzej Duda and Tadeusz Czachórski. 1987. Performance evaluation of fork and join synchronization primitives. *Acta Informatica* 24 (1987), 525–553.

[12] Greg Franks and Murray Woodside. 1998. Performance of multi-level client-server systems with parallel service operations. In *Proceedings of the 1st international workshop on Software and performance*. 120–130.

[13] Heidelberger and Trivedi. 1983. Analytic queueing models for programs with internal concurrency. *IEEE Trans. Comput.* 100, 1 (1983), 73–82.

[14] Patricia A Jacobson and Edward D Lazowska. 1982. Analyzing queueing networks with simultaneous resource possession. *Commun. ACM* 25, 2 (1982), 142–151.

[15] Gokcen Kestor, Sriram Krishnamoorthy, and Wenjing Ma. 2017. Localized fault recovery for nested fork-join programs. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 397–408.

[16] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. 1984. *Quantitative system performance: computer system analysis using queueing network models.* Prentice-Hall, Inc.

[17] John DC Little. 1961. A proof for the queuing formula: L= λ W. *Operations research* 9, 3 (1961), 383–387.

[18] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. 1995. Specifying distributed software architectures. In *Software Engineering—ESEC'95: 5th European Software Engineering Conference Sitges, Spain, September 25–28, 1995 Proceedings 5*. Springer, 137–153.

[19] Victor W Mak and Stephen F. Lundstrom. 1990. Predicting performance of parallel computations. *IEEE Transactions on Parallel & Distributed Systems* 1, 03 (1990), 257–270.

[20] Randolph Nelson and Asser N Tantawi. 1988. Approximate analysis of fork/join synchronization in parallel queues. *IEEE transactions on computers* 37, 6 (1988), 739–743.

[21] Martin Reiser and Stephen S Lavenberg. 1980. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM (JACM)* 27, 2 (1980), 313–322.

[22] S Salza and SS Lavenberg. 1981. Approximating response time distributions in closed queueing network models of computer performance. (1981).

[23] Alexander Thomasian. 2014. Analysis of fork/join and related queueing systems. *ACM Computing Surveys (CSUR)* 47, 2 (2014), 1–71.

[24] Kishor S Trivedi. 2008. *Probability & statistics with reliability, queuing and computer science applications.* John Wiley & Sons.

[25] Elizabeth Varki. 1999. Mean value technique for closed fork-join networks. *ACM SIGMETRICS Performance Evaluation Review* 27, 1 (1999), 103–112.