Journal of Circuits, Systems, and Computers

APEM - Approximate Performance Evaluation for Multi-core Computers

Lei Zhang

Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, ON L8S4K1, Canada

Douglas G. Down

Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, ON L8S4K1, Canada

Mean Value Analysis (MVA) has long been a standard approach for performance analysis of computer systems. While the exact load-dependent MVA algorithm is an efficient technique for computer system performance modelling, it fails to address multi-core computer systems with Dynamic Frequency Scaling (DFS). In addition, the load-dependent MVA algorithm suffers from numerical difficulties under heavy load conditions. The goal of our paper is to find an efficient and robust method which is easy to use in practice and is also accurate for performance prediction for multi-core platforms. The proposed method, called APEM[a], uses a flow-equivalent performance model designed specifically to address multi-core computer systems and identify the influence on the CPU demand of the effect of DFS. We adopt an approximation technique to estimate resource demands to parameterize MVA algorithms. To validate the application of our method, we investigate three case studies with extended TPC-W benchmark kits, showing that our method achieves better accuracy compared with other commonly used MVA algorithms. We compare the three different performance models, and we also extend our approach to multi-class models.

Keywords: performance evaluation; product-form; mean value analysis; dynamic frequency scaling

1. Introduction

The exact Mean Value Analysis algorithm for the analysis of closed queueing networks with product-form steady-state distribution was introduced by Lavenberg and Reiser.[1] Compared with other methods that obtain the steady-state distribution by solving a (large) number of linear equations, the exact MVA algorithm is a simple recursion, which derives mean performance metrics with any number of jobs from underlying service demands. Besides networks with constant service rates, the MVA algorithm has been used for queueing networks containing load-dependent nodes - nodes at which the service rate varies with the number of jobs present (the algorithm can be found in Chapter 14 of Menascé et al.'s work).[2]

[a]APEM stands for Approximate Performance Evaluation for Multi-core computers

The motivation for this paper is derived from our experiences with implementations of MVA-based performance models. While the MVA algorithm itself is simple, its applicability can be challenged by features introduced in modern computer systems. Such features can violate product-form assumptions. When the steady-state distribution is not product-form, the MVA algorithm is infeasible in such cases. Another challenge comes from systems employing Dynamic Frequency Scaling (DFS) techniques.

DFS is a technique employed by operating systems where a processor is running at a frequency less than its maximum in order to conserve power. In Linux operating systems, DFS is supported by the "ondemand" policy.[3] The "ondemand" governor sets the CPU frequency depending on the current usage. The operating system checks the load/utilization regularly (according to a given sampling rate). When the load rises above a given threshold value, Linux sets the CPU to run at the highest frequency. On the other hand, when the load falls below the same threshold, Linux sets the CPU to run at the next lowest frequency that maintains the load below the threshold. In particular, the CPU frequency is scaled down to its minimum if the CPU is idle. Subsequently, processors become faster (or the CPU demand decreases) when the load increases in the system. In addition, each processor's frequency is independently controlled by the operating system. Not only does this make a load-independent performance model inapplicable, but it also makes the parameterization for a load-dependent performance model more complicated.

DFS is well studied in the literature in terms of its effects on the tradeoff between power consumption and performance. Recent research has focused on either determing the minimum frequency (given job deadlines), or the best range of frequencies (given energy budgets), or hybrid models for both. However, to the best of our knowledge, there is no research that directly examines DFS's effects on MVA algorithms. Menascé examined the tradeoffs between CPU clock frequency and system performance.[4] His paper presented a mechanism to adjust automatically the minimum CPU frequency according to the workload intensity and the maximum average response time desired in corresponding Service Level Agreements (SLAs). Dhiman and Rosing proposed a scaling technique using an online learning algorithm to select the best range of frequencies in accordance with task requirements in multi-tasking environments.[5] Wierman et al. studied two scaling techniques - dynamic speed scaling and static processing - to address both worst case and average case performance with a processor sharing scheduler, and built a bridge between them.[6,7]

In this paper, we present a new method for DFS-enabled multi-core platforms that cannot be modelled as product-form queueing networks. We call this method APEM - Approximate Performance Evaluation for Multi-core computers. We first introduced the APEM method for closed queueing networks.[8] Here, we extend its applicability to open and semi-open queueing networks. The APEM method involves flow-equivalent aggregation to build an approximate product-form network for a non-product-form system. Then, the APEM method applies an appropriate

MVA algorithm as the means to determine the performance of the approximating network. A service demand approximation is embedded in the MVA algorithm in order to address the effects of DFS.

The main contributions of APEM are as follows:

(i) The APEM method focuses on non-product-form queueing networks that are applicable for analytical performance modelling for modern computer systems.
(ii) The APEM method is the first performance solution which addresses service demand estimation for DFS.
(iii) The APEM method is applicable for closed, open, and semi-open queueing networks.
(iv) The APEM method deploys new MVA algorithms with DFS adjustments in both single-class and multi-class models for different types of queueing networks.

Table 1: Notation

M	number of resources (devices)
N	number of users
λ	arrival rate
Z	average think time
D_m	resource demand at the mth resource (e.g., CPU and disk)
μ_m	service rate at the mth resource
K_m	number of servers at the mth resource
R	average response time
X	throughput
Q	average queue length
$P_m(j)$	the probability that j users are present at the mth resource

The remainder of the paper is organized as follows. Section 2 provides the details of APEM. Section 3 presents three case studies to verify the accuracy of APEM. Section 4 compares and discusses the differences among the three queueing models in those case studies. We extend our proposed algorithms to multi-class systems in Section 5. Section 6 reviews related work. Finally, a summary is provided in Section 7. For reference, the notation presented in Table 1 is used in the remainder of the paper.

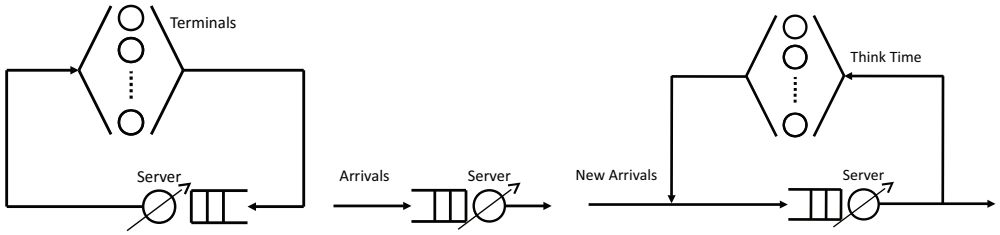
2. APEM

In this section, the APEM method is discussed. Firstly, we introduce the possible queueing networks that can be used. Secondly, we describe the impact of DFS-

enabled multi-core systems, and then propose our solution. Thirdly, three approximate MVA algorithms with DFS adjustment are introduced. Finally, we present the APEM method in detail.

2.1. *Queueing Networks*

Employing queueing models for modern computer systems can lead to non-product-form network models. For example, simultaneous resource possession, locking behaviours in database servers, priority scheduling, high service demand variability, and process synchronization, all common on modern computers, violate conditions for product-form solutions.[2] Thus, approximate techniques, such as flow-equivalent aggregation,[9] have been developed. Generally speaking, flow-equivalent aggregation reduces a non-product-form queueing network to an approximate product-form queueing network with a Flow-Equivalent Server (FES). The FES is a load-dependent node. The service rates of the FES are equal to the throughputs of the original network with n jobs present (n ranges from 1 to N). The throughputs can be calculated from transaction logs.



(a) Closed system (b) Open system (c) Semi-open system

Fig. 1: Closed, open, and semi-open queueing networks

The flow-equivalent aggregation method can be used for closed, open, and semi-open system models. Figures 1a, 1b and 1c illustrate three examples of those systems, respectively.

A closed queueing system with a load-dependent server is shown in Figure 1a. It is assumed that the number of users is fixed in a closed system. In the system, a user triggers requests to the server. After one request is completed, the user then waits for a think time, with mean Z. Another request is then submitted to the server. This cycle is repeated indefinitely. Generally speaking, closed systems have fixed user population and load-dependent arrival rate. In real systems, the user population is rarely constant. However, closed models are commonly deployed when system designers want to evaluate the performance for a system under stressful workloads.

Figure 1b shows an open queueing system with a load-dependent server. Open systems do not model the size of the user population, but focus on request level

performance. New requests arrive with rate λ. That means new arrivals do not rely on the completions of previous requests. Generally, open systems have fixed arrival rate and thus the number of requests in the system varies over time. In real systems, the arrival rate is frequently time sensitive, for example, an e-commerce website may expect more customers during weekends than during weekdays. However, open queueing networks can be used to evaluate the performance for systems if one can consider the arrival rate to be fixed over particular time intervals.

Semi-open systems have the features of both open and closed systems. Figure 1c depicts a semi-open queueing system with a load-dependent queue. In semi-open systems, users arrive according to an arrival rate λ. Once a user enters the system, it will send a sequence of requests. After the completion of a request, a user waits for a think time that has mean Z, and then sends the next request. The expected number of requests per user is T. A user leaves the system once all of its requests are completed. In a semi-open system, users arrive to the system as an open system; once in the system, users send requests as in a closed system. Semi-open systems can be considered as generalizations of both open and closed systems. When $\lambda = 0$ and $T = \infty$ (no jobs arrive from outside or depart from the system), the resulting system is a closed system. When the number of requests associated with each user is exactly one, the result is an open system. While a semi-open queueing network gives more flexibility to model real systems, it is more difficult to analyze. Note that both open and semi-open systems can have restricted capacity, which can make the analysis more complex. A queueing network with restricted capacity has finite queues. After the maximal capacity has been achieved, new arrivals will be blocked or rejected. In this paper, we assume that our open and semi-open systems do not have restricted capacities.

2.2. *Service Demand Estimation*

To estimate service demands for DFS-enabled computers, we need to take the effects of DFS into account. Consider a scenario where a core is idle for a sufficient length of time such that its frequency has been scaled down to the minimum to save energy. Upon a job arrival, the core is still at this power-saving stage and needs some time to "warm up". Consequently, the service demand of the job is increased due to the decreased frequency during this interval. The parameterization of the MVA algorithm can be effectively enhanced with a corresponding service demand adjustment.

Formally, we state the following observation for a DFS-enabled computer: *There exists a probability that a job will be (partially) processed at a slow rate, if the core where it is allocated has been idle for a sufficient interval and its frequency has thus been scaled down to the lowest value.*

Based on this observation, our next step is to estimate this probability. On a DFS-enabled platform, the Linux operating system takes workload (utilization) samples every 10 milliseconds under the "ondemand" policy, and then decides

whether to scale the core frequency up or down. Then the probability that an arriving job will enter a server working at the slowest possible rate is identical to the probability that the time interval between a job arrival to an idle server and the last departure is larger than the sampling interval. The probability is given by $P(\mathcal{A} > s)$, where $\mathcal{A}$ is an interarrival time and s is the sampling interval. In the queueing model, we assume that jobs arrive with the interarrival time (or think time) following an exponential distribution with mean Z. We let $P^{slow}(n)$ be the probability that a job arrives to a slow server given that there are n jobs in the system. By the memoryless property, we can write

$$P^{slow}(n) = P(\mathcal{A} > s|n) \approx Exp(-\frac{sn}{ZK}),$$

where n is the number of jobs in the system, and K is the number of cores in the CPU. Here we assume that each core has the same arrival rate.

Using this, we have the adjustment of the CPU service demand as follows,

$$D(i) = D^{unadjusted}(i)\big(1 - P^{slow}(n)\big) + D^{slow}(i)P^{slow}(n), \tag{1}$$

where $D^{unadjusted}(i)$ is the CPU demand with i jobs at the CPU, and $D^{slow}(i)$ is the service demand of the CPU with one core running at the minimum frequency. To calculate $D^{slow}(i)$, it is easier to compute the inverse of the service rate - $\mu^{slow}(i)$ - using the following equation:

$$\mu^{slow}(i) = \mu^{slow}(1) + \mu(i-1), \tag{2}$$

where $i = 1, \ldots, K$, $\mu^{slow}(1)$ is measured with one job in the system and the CPU running at the minimum frequency, and $\mu(i-1)$ is the service rate with one less job at the CPU.

Formally, we make the following assumptions to develop our estimate:

(i) The interarrival times follow an exponential distribution.
(ii) If an arrival enters a "slow" server, then the entire job will be processed at the slowest possible rate.
(iii) Each logical core works independently.
(iv) The mean think time is sufficiently large.

The proposed technique is a somewhat crude means to adjust the service demand due to the effects of DFS. The real "ondemand" DFS and multi-core CPU scheduling policies within Linux are much more complicated. In reality, if the job's processing time is longer than the sampling interval, only part of the job will be processed at the slowest possible rate. The core's frequency will be scaled up at the next sampling point. Thus, the remaining part of the job will be processed at higher rates. As a result, this approximation will be more effective when the CPU demand is small. In addition, in a multi-core processor, one job running on a core may be impacted by another job running on a different core due to shared memory interference. Although our technique is a coarse approximation, it is to the best of

our knowledge the first attempt to account for the effects of DFS in determining demands.

2.3. *Approximate MVA Algorithms for Different Performance Models*

The service demand estimation needs to be embedded into the corresponding MVA algorithms. In this section, we will present three different MVA algorithms for closed, open and semi-open systems, respectively.

2.3.1. *Approximate MVA Algorithm for Closed Systems*

The original MVA algorithm for load-dependent queues in closed systems suffers from numerical instability issues. It may exhibit numerical difficulties under heavy load conditions which eventually result in unreasonable results, such as negative throughputs, response times and queue lengths. In order to address the numerical instability of the load-dependent MVA algorithm, we adopt the Conditional MVA (CMVA) algorithm,[10] which relates the average queue length of a load-dependent server to the conditional queue length as seen by a job during its residence time at that server.

Algorithm 1 presents the CMVA algorithm with the service demand adjustment for DFS. We call this algorithm CMVA-DFS. Unlike the original CMVA algorithm, we assume that all of the resources are load dependent, because we only have FESs in this case study. This extension of the CMVA-DFS algorithm is straightforward.

2.3.2. *Approximate MVA Algorithm for Open Systems*

For the MVA algorithm in the closed model, we do not make any assumptions on service demands. In the open model, we make the assumption that the service demand eventually becomes constant as the system load increases. This means that there exists a finite $\bar{N}$ such that $D_m(n) = D_m(\bar{N})$ if $n > \bar{N}$. This assumption makes physical sense. Firstly, suppose that the value of service demand approaches zero as the system load increases, i.e., there is no limit to how fast the corresponding resource can become. This scenario is physically impossible. Secondly, the service demand going to infinity (as workload increases) would automatically lead to an unstable system. One remedy would be to control the number of jobs in the system. For example, this assumption automatically holds in an open system with restricted capacity. Thirdly, this assumption could be an approximate solution for the case that the service demand converges, for example, $D(n) = 1 + 1/n$. Here, $D(n)$ has a limiting value of one, but it never reaches one. However, the open MVA algorithm may encounter numerical issues (e.g., overflow) if the service demand converges very slowly to its limit. We will see the reason behind this later, and more concrete examples will be analyzed in Section 3.3. Last but not least, this assumption is

Algorithm 1 The CMVA-DFS algorithm

Input:
$Z, M, N, D_m^{unadjusted}(n), D_m^{slow}(n), s, K_m$
Output:
Q, X, R
Initialization:
for $m = 1 \to M$ and $t = 1 \to N + 1$ **do** $Q_m(0, t) = 0$
  DFS adjustment:
  for $m = 1 \to M$ **do**
    if m is a DFS-enabled resource **then**
      for $n = 1 \to N$ **do**
        $P_m^{slow}(n) = Exp(-sn/ZK_m)$
        $D_m(n) = D_m^{unadjusted}(n)\big(1 - P_m^{slow}(n)\big) + D_m^{slow}(n)P_m^{slow}(n)$
      end for
    end if
  end for
  Iteration:
  for $n = 1 \to N$ **do**
    for $t = 1 \to N - n + 1$ **do**
      for $m = 1 \to M$ **do**
        if $n == 1$ **then** $D_m(n, t) = D(t)$
        else $D_m(n, t) = D_m(n - 1, t)X(n - 1, t)/X(n - 1, t + 1)$
        end if
      end for
      for $m = 1 \to M$ **do**
        $R_m(n, t) = D_m(n, t)\big(1 + Q_m(n - 1, t + 1)\big)$
      end for
      $X(n, t) = n/\big(Z + \sum_{m=1}^{M} R_m(n, t)\big)$
      for $m = 1 \to M$ **do**
        $Q_m(n, t) = X(n, t)R_m(n, t)$
      end for
    end for
  end for

supported by our tests - we observed that the service rate of the FES eventually becomes roughly a constant when the system load is sufficiently large.

Algorithm 2 presents the open MVA algorithm for load-dependent queues and DFS-enabled servers. We call this algorithm Open MVA-DFS. The open MVA algorithm requires a stability condition: $U_m < \alpha_m(\bar{N}_m)$, where $\alpha_m(\bar{N}_m)$ is a service demand ratio defined as $\alpha_m(\bar{N}_m) = D_m(1)/D_m(\bar{N}_m)$. Our algorithm is based on the MVA algorithm introduced by Menascé et al. (Chapter 14),[2] but has two differences:

Algorithm 2 The Open MVA-DFS algorithm

Input:

$\lambda, M, K_m, D_m^{unadjusted}(n), D_m^{slow}(n), s, \bar{N}_m$

Output:

Q, R

Initialization:

DFS adjustment:
for $m = 1 \rightarrow M$ **do**
  if m is a DFS-enabled resource **then**
    for $n = 1 \rightarrow \bar{N}_m$ **do**
      $P_m^{slow}(n) = Exp(-sn\lambda/K_m)$
      $D_m(n) = D_m^{unadjusted}(n)\big(1 - P_m^{slow}(n)\big) + D_m^{slow}(n)P_m^{slow}(n)$
    end for
  end if
end for
for $m = 1 \rightarrow M$ **do**
  $U_m = \lambda D_m(1)$
  for $n = 1 \rightarrow N$ **do**
    $\alpha_m(n) = D_m(1)/D_m(n)$
  end for
  if $U_m \geq \alpha_m(\bar{N}_m)$ **then**
    Stop and exit
  end if
end for
Iteration:
for $m = 1 \rightarrow M$ **do**

$$P_m(0) = \Big[\sum_{j=0}^{\bar{N}_m} \prod_{k=0}^{j} \frac{U_m}{\alpha_m(k)} + \prod_{j=1}^{\bar{N}_m} \frac{\alpha_m(\bar{N}_m)}{\alpha_m(j)} \times \frac{\big(U_m/\alpha_m(\bar{N}_m)\big)^{\bar{N}_m+1}}{1-U_m/\alpha_m(\bar{N}_m)}\Big]^{-1}$$

$$Q_m = P_m(0) \times \Big\{\sum_{j=1}^{\bar{N}_m} j \prod_{k=1}^{j} \frac{U_m}{\alpha_m(k)} + \prod_{j=1}^{\bar{N}_m} \frac{U_m}{\alpha_m(j)} \times \frac{U_m}{\alpha_m(\bar{N}_m)} \times \Big[\frac{U_m/\alpha_m(\bar{N}_m)+(\bar{N}_m+1)\big(1-U_m/\alpha_m(\bar{N}_m)\big)}{\big(1-U_m/\alpha_m(\bar{N}_m)\big)^2}\Big]\Big\}$$

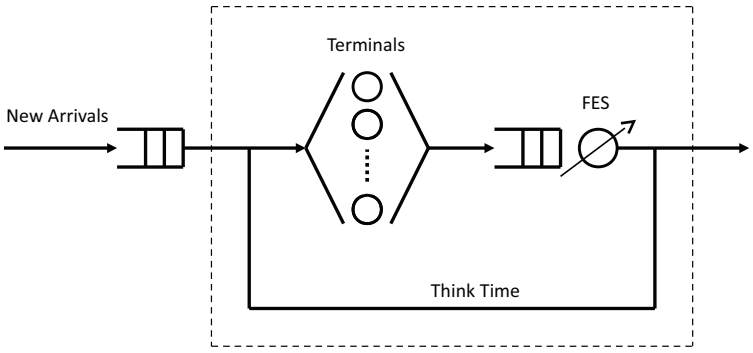
  $R_m = Q_m/\lambda$
end for

(i) The effect of DFS on the service demand is considered. Note that the estimate here is slightly different from the estimate in Algorithm 1. Instead of using the mean think time Z, we use the arrival rate λ. Since the number of jobs varies in an open system, the calculation of $P_m^{slow}(n)$ can be potentially inaccurate. One approach that we can choose to bound the value of $P_m^{slow}(n)$ is to check the condition $P_m^{slow}(n) \geq P_m(0)$. If this condition is violated, the estimation of $P_m^{slow}(n)$ must have accuracy issues. During our experiments, this condition has never been violated, although it is not clear how to modify the value of

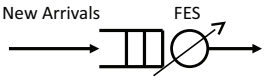
$P_m^{slow}(n)$ if the condition is violated. This would be a useful avenue for future work.

(ii) When calculating the value of $P_m(0)$, we have a term $\prod_{j=1}^{\bar{N}_m} \big(\alpha_m(\bar{N}_m)/\alpha_m(j)\big)$ in our algorithm. In the original algorithm, this term is presented as $\big(\alpha_m(\bar{N}_m)\big)^{\bar{N}_m}/\beta_m(\bar{N}_m)$, where $\beta_m(\bar{N}_m) = \alpha_m(1)\times\cdots\times\alpha_m(\bar{N}_m)$. However, the value of $\beta_m(\bar{N}_m)$ can potentially cause numerical issues if $\bar{N}_m$ is very large. To address this, we break down $\beta_m(n_m)$ and perform divisions of $\alpha_m(n_m)$ before the multiplication in our algorithm. However, overflow or underflow could still occur if the service demand converges very slowly to its limit (as mentioned before). On the one hand, the product of $\alpha_m(\bar{N}_m)/\alpha_m(j)$ can cause overflow if $\alpha_m(\bar{N}_m)/\alpha_m(j)$ is larger than one (e.g., $\prod_n(1+1/n), n = 1,\ldots,\infty$). In this case, $D_m(j)$ is larger than $D_m(\bar{N}_m)$, and it converges to $D_m(\bar{N}_m)$ as j increases. On the other hand, the product of $\alpha_m(\bar{N}_m)/\alpha_m(j)$ can be rounded to zero (underflow) if $\alpha_m(\bar{N}_m)/\alpha_m(j)$ is smaller than one (e.g., $\prod_n[1-1/(n+1)], n = 1,\ldots,\infty$). In this case, $D_m(j)$ is smaller than $D_m(\bar{N}_m)$, and it converges to $D_m(\bar{N}_m)$.

2.3.3. *Performance Model for Semi-open Systems*



(a) Lower level network



(b) Upper level network

Fig. 2: A hierarchical model with two levels of FES

Algorithm 3 The Open MVA algorithm for semi-open systems

Input:

$\lambda, M, D_m^r(n), \bar{N}_m, T, Z$

Output:

X^r, R^r

Initialization:

for $m = 1 \to M$ **do**
  for $n = 1 \to N$ **do**
    $D_m(n) = T \times D_m^r(n)$
    $\alpha_m(n) = D_m(1)/D_m(n)$
  end for
  $U_m = \lambda D_m(1)$
  if $U_m \geq \alpha_m(\bar{N}_m)$ **then**
    Stop and exit
  end if
end for

Iteration:

for $m = 1 \to M$ **do**

$$P_m(0) = \Big[\sum_{j=0}^{\bar{N}_m} \prod_{k=0}^{j} \frac{U_m}{\alpha_m(k)} + \prod_{j=1}^{\bar{N}_m} \frac{\alpha_m(\bar{N}_m)}{\alpha_m(j)} \times \frac{\big(U_m/\alpha_m(\bar{N}_m)\big)^{\bar{N}_m+1}}{1-U_m/\alpha_m(\bar{N}_m)} \Big]^{-1}$$

$$Q_m = P_m(0) \times \{ \sum_{j=1}^{\bar{N}_m} j \prod_{k=1}^{j} \frac{U_m}{\alpha_m(k)} + \prod_{j=1}^{\bar{N}_m} \frac{U_m}{\alpha_m(j)} \times \frac{U_m}{\alpha_m(\bar{N}_m)} \times \Big[\frac{U_m/\alpha_m(\bar{N}_m) + (\bar{N}_m+1)\big(1-U_m/\alpha_m(\bar{N}_m)\big)}{\big(1-U_m/\alpha_m(\bar{N}_m)\big)^2} \Big] \}$$

$R_m = Q_m/\lambda$

end for

for $m = 1 \to M$ **do**
  $R_m^r = (R_m - T \times Z)/T$ ▷ response time in seconds per request
end for

$X^r = \lambda T$ ▷ throughput in requests per second

Semi-open systems are much more complex to analyze than closed or open systems, and we cannot use MVA algorithms directly. For semi-open systems, we use hierarchical modelling to solve the network.[11] The FES technique is the key to hierarchical modelling. The top level of the model may consist of one or more FESs. An FES is represented as a more detailed subsystem in the next level down. This lower subsystem may also contain one or more FESs, which can again be represented in detail in the next level down, and so forth. The level of hierarchy grows until the lowest level models can be solved. In general, service rates of an FES in an upper level are obtained from solving the corresponding subsystem in the lower level.

We model the system as a non-product-form queueing network shown as in Figure 2a. The system has two levels of FES. The lower level network is a closed

system with an FES which represents all of the resources (CPUs and disk) in the system. The lower level is shown within the dashed lines in Figure 2a. Then, the entire lower level closed network is replaced by an FES. We build the upper level open network as in Figure 2b. The service rate curves of the FES in the lower level with n jobs present are also obtained by the throughput of the system with different numbers of jobs. The service rate curves of the FES in the upper level are obtained by the throughput of the system with various numbers of jobs and think time with mean Z.

We first use Algorithm 1 for closed systems to calculate the throughputs with n jobs and mean think time Z. The throughputs are the service rates of the FES in the upper level, so that we use their inverses - $D_m^r(n)$ - as input parameters for Algorithm 3. Other input parameters are obtained directly from the system. Finally, we use Algorithm 3 to solve the upper level open network, which is equivalent to the entire semi-open system.

Algorithm 3 is also based on the MVA algorithm introduced by Menascé et al. (Chapter 14).[2] One difference is that we need to calculate the expected number of requests per user T in Algorithm 3. The performance metrics in Algorithm 1 are at request level. However, the performance metrics are at user session level in Algorithm 3, because the unit of the input parameter λ is users per second. As a result, we need $D_m(n) = T \times D_m^r(n)$ to transform from the service demand of a request $D_m^r(n)$ to the service demand of a user $D_m(n)$. Finally, the outputs of Algorithm 3 are all transformed to request level.

2.4. *The APEM Method*

Given the performance models introduced, we are now able to provide a detailed description of the APEM method.

(i) **Characterize the system workload structure**. In order to accurately model the system performance, workload characterization is the necessary first step, whether real workloads or benchmarks are employed. Firstly, representative workloads are identified. Secondly, the representative workloads are classified by different resource intensities, for example, a CPU-intensive workload is separated from an I/O-intensive workload. Thirdly, special features are characterized, e.g., simultaneous resource possessions or locking behaviours. Fourthly, all representative workloads should have transaction logs. Last but not least, job structure is also important for performance models. A job structure consists of job and request relationships, e.g., whether jobs are independent of each other, or if a job triggers more than one request. For example, we employ the TPC-W benchmark in our case studies in Section 3, for which all of the workload classes and job structures are characterized in its specification.[12]

(ii) **Determine the performance model**. Based on the characterized workload and job structure, an appropriate performance model is chosen. If any features that violate conditions for product-form solutions (in Section 2.1) are found

in Step 1, a non-product-form queueing network should be considered as a candidate. If a system has a fixed user population, then a closed queueing network is an appropriate choice. If a system has a variable user population, then an open queueing network is a good fit. If a system has features of both open and closed systems, a more complex model should be considered, i.e., a semi-open queueing network.

(iii) **Obtain the service rates**. Service rates or service demands are important input parameters for MVA algorithms. There are many approaches to obtain service rates, all of which involve system measurements. The Service Demand Law is one of the most popular approaches. Since the resource utilization and the resource demand are assumed to have a linear relationship, regression techniques can be employed when measurements of utilizations or throughput are not available.[13,14,15,16,17] In addition, the response time is related to the resource demand in a nonlinear manner. As a result, different estimation techniques can be used to estimate the resource demand from response time measurements.[18,19] If the FES technique is involved, the service rates are obtained by measuring throughputs of the appropriate aggregated system with different numbers of jobs. If the system is DFS enabled, the service rates of a slow server are obtained as discussed in Section 2.2.

(iv) **Choose an MVA algorithm**. Corresponding to the performance model(s) chosen in Step 2, we choose a corresponding MVA algorithm(s) to solve the performance model. Different MVA algorithms fit different scenarios. Closed MVA algorithms are used to solve closed systems, while open MVA algorithms are used for open systems. Single-class MVA algorithms are easy to parameterize, and multi-class MVA algorithms are deployed for detailed class-level performance evaluation. While MVA algorithms with load-independent queues work with constant service demands, MVA algorithms with load-dependent queues work for the FES technique. When it becomes problematic to use an exact MVA algorithm, an approximate MVA algorithm can be chosen. When solving a closed queueing network with load-dependent queues, it is critical to choose a numerically stable MVA algorithm.

(v) **Solve the performance model**. The appropriate MVA algorithm is used to obtain the desired performance metrics, such as the mean response times, the mean number of jobs in the system, and the throughputs. The service rates in Step 3 are used as input parameters. Other input parameters, such as the mean think time, the arrival rate, and the user population, may also be required.

(vi) **Analyze the results**. The results from the MVA algorithm are compared with system measurements. If the results are sufficiently close, the performance model is validated. Otherwise, we need to review all previous steps, and identify any potential issues which caused the failure. Errors in the performance model should be analyzed, with particular attention to which scenarios give rise to the largest errors. As a rule of thumb, response time errors within 20% are considered acceptable (see Chapter 4 in the work of Menascé et al.).[2]

(vii) **Predict the performance of the system**. The goal of performance modelling is to evaluate the system performance without extensive operation of the real system. A performance model can be used to evaluate different workloads on the system by adjustments of input parameters, such as the number of jobs, the think time, the arrival rate, and the service demand. Designed workloads should reflect user behaviours when the system is operational. For example, the model can be used to predict the performance of the system with peak workload. Compared to stress testing solutions, the performance model approach can be both inexpensive and efficient.

3. Accuracy of APEM with Different Performance Models

In this section, we demonstrate the use of the APEM method for closed, open, and semi-open systems, respectively.

All three case studies in this section are built on the same testbed, a Dell desktop computer equipped with an Intel i7-2600 quad-core processor (octo-core logically with Hyper-Threading Technology), 8GB RAM, and a 1TB disk (7200 RPM). The processor is DFS enabled, and has frequency range from 1.6 GHz to 3.4 GHz. For the operating system, we used Ubuntu Server 12.04.3 LTS (kernel version 3.5.0-44). The testbed carries a web application server and a database. The web server accepts client requests and issues queries to the database in order to retrieve the requested data. For the web server and the database server, we used JBoss 3.2.7 and MySQL 5.1.70, respectively.[20,21] The average response time is our performance metric of interest.

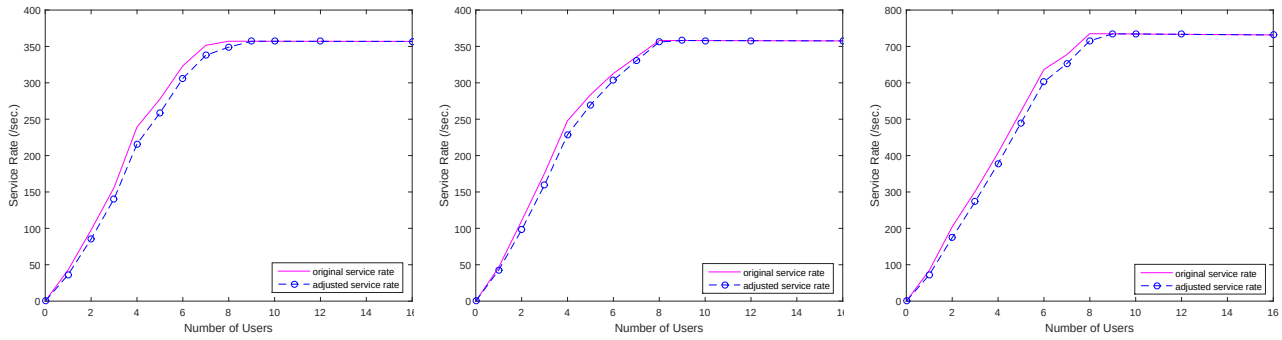
3.1. *Experiments for Closed Systems*

The TPC-W that we use is the implementation of Horvath.[22] The number of users ranges from 1 to 800, and we set the average think time (Z) to 3.5 seconds, in order to validate APEM under different workloads. In our tests, the CPU utilization varies from 1.09% to 61.59%, and the average CPU frequency varies from 1.61 GHz to 2.61 GHz.

We apply APEM in the following manner:

(i) Model the non-product-form system as a closed queueing network with an FES.
(ii) Calculate the service demands of the FES under different workload mixes. This step involves measurements of the throughputs (service rates) with various numbers of users in the FES.
(iii) Calculate the service demands of the FES with slow cores under different workload mixes. This step involves measurements of the throughput with one user in the FES while the CPU is running at its minimum frequency.
(iv) Use the CMVA-DFS algorithm to predict average response times for the system under different workload mixes.

(v) Measure the average response times of the system under different workload mixes. This step involves measurements of the average response times with the corresponding number of users from the previous step.
(vi) Validate predicted results with corresponding measurements under different workload mixes.



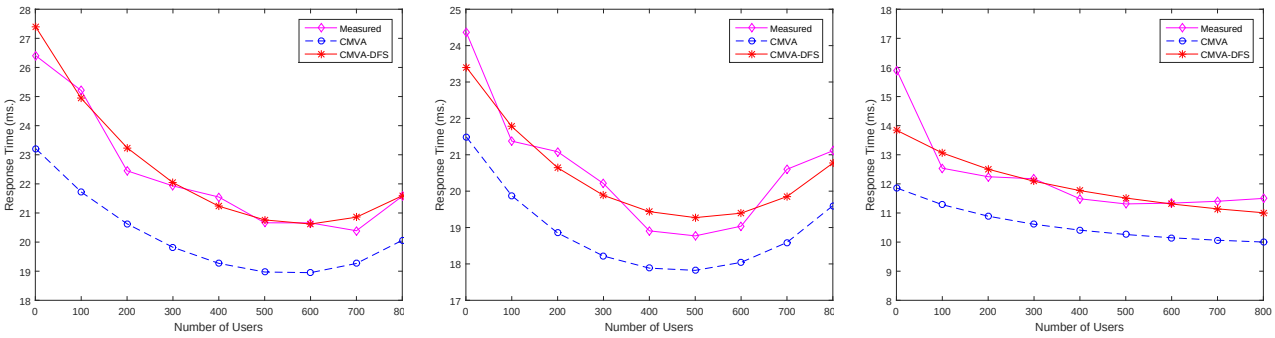
(a) Browsing mix (b) Shopping mix (c) Ordering mix

Fig. 3: Service rate of FES with different mixes

For the second step, we need to generate a set of service demands with varying numbers of users, from one user to a potentially very large number of users. To accomplish this, we select several values for the number of users, calculate the service rates from measurements (measured throughputs), then interpolate between these points to generate a service rate curve as shown in Figure 3. The points on the original service rate curve are calculated from measurements (the service rate is equal to the measured throughput). Other points on the curve are approximated by performing linear interpolation between the measured points. For instance, the measured service rates correspond to the number of users - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12 and 16 in Figure 3. Note that this approximation may affect the accuracy of the performance model, but we can increase its accuracy by increasing the number of points which are calculated from measurements. The balance between the time consumed in measurements/parameterizing the model and the model accuracy can be adjusted as necessary. Further discussion based on experimental results with different granularities can be found in our previous work.[8]

For the third step, we calculate the adjusted service demands using Eq. (1) in Section 2.2. The service rates of cores running at the minimum frequency can be measured by fixing the CPU frequency at its minimum, or the mean think time at a large value (to ensure a CPU has sufficient idle time to scale down its frequency to the minimum). Then, Eq. (2) is used to calculate the desired service rates. These

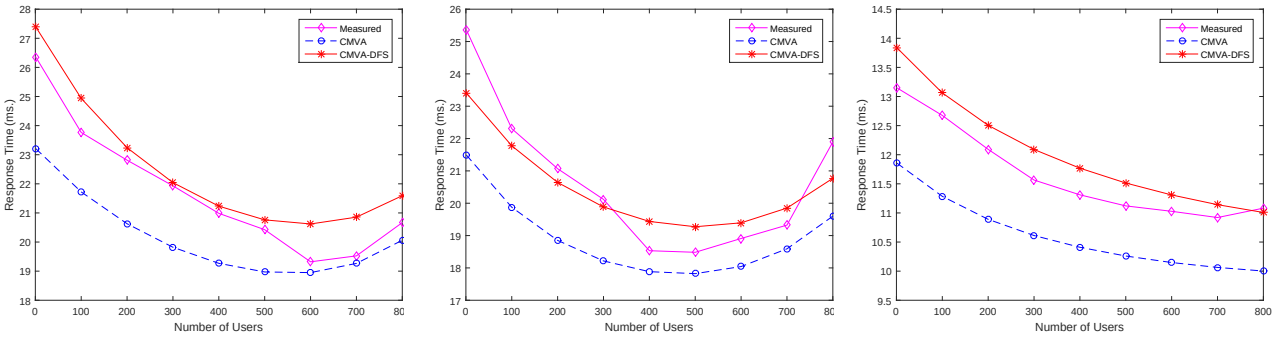
curves are also shown in Figure 3. Note that these service rate curves will be reused for the parameterization in the next two case studies.



(a) Browsing mix (b) Shopping mix (c) Ordering mix

Fig. 4: Predicted and measured R with different mixes and $Z = 3.5$ seconds

Finally, the experimental results are shown in Figure 4. The results from APEM are compared with the results from measurements, and the original CMVA algorithm. As can be seen in Figure 4, compared with the CMVA algorithm, which consistently underestimates the average response time (by as much as 25.41% for the ordering mix), the CMVA-DFS algorithm produces better predictions. The underestimation issue of the CMVA algorithm is caused by the overestimation of the service rates (as shown in Figure 3).



(a) Browsing mix (b) Shopping mix (c) Ordering mix

Fig. 5: Predicted and measured R with different mixes and $Z = 3.5$ seconds (no JBoss logging)

A small number of abnormally large response times are observed during our experiments - these values have a significant effect on the results, especially when the system is under heavy load. After extensive experimentation and analysis, we determined that these outliers arise due to JBoss logging. We have two options to remove those effects: we can simply disable JBoss logging, or adopt a univariate method to detect and remove outliers. In this paper, we choose the latter approach, because one may prefer to keep JBoss logging enabled for administrative purposes. Here, we show another set of results in Figure 5 with JBoss logging disabled to demonstrate that our method also works in such a case. As can be seen in Figure 5, the CMVA-DFS algorithm still produces better predictions compared to the CMVA algorithm. More discussion and results with JBoss logging disabled can be found in my thesis.[23] The outlier detection technique is discussed in Appendix A.

3.2. *Experiments for Open Systems*

The original TPC-W benchmark assumes a closed system. The number of jobs in the system is fixed, and each job triggers a sequence of requests. For modelling an open system, we modify the TPC-W kit introduced in Section 3.1 so that new requests arrive to the system following a Poisson process. The original TPC-W kit has a main thread which initializes all user sessions (EBs). Each user session is an independent thread that sends a sequence of requests until the end of the test. After receiving the reply for the last request, a user session will wait for a think time with mean Z and send the next request.

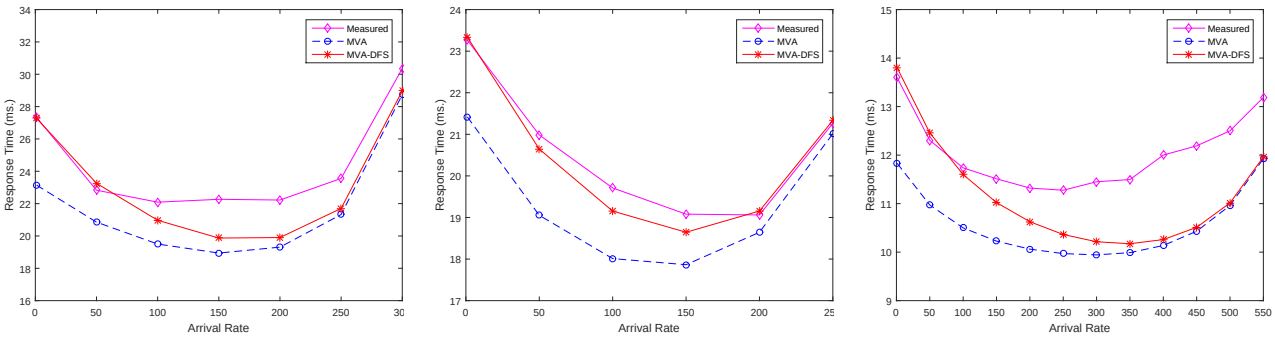
We modified the TPC-W kit for open systems. In that TPC-W kit, the main thread is the only one that decides when to send a request to the system. It sleeps for an exponentially distributed interarrival time. After that, the main thread checks if there are idling user threads waiting. If yes, one of these users receives a signal from the main thread, and makes exactly one request. Otherwise, the main thread creates a new user thread, and the new thread immediately makes one request. On the user thread side, a user thread blocks and waits for the signal from the main thread after sending each request. The source code of this modified TPC-W kit can be acquired from our website.[24] To verify our method under different workloads, we vary the arrival rate from 1 per second to 550 per second.

We apply APEM in the following manner:

(i) Model the system as an open queueing network with an FES.
(ii) Calculate the service demands of the FES under different workload mixes. This step involves measurements of the throughputs (service rates) with various numbers of requests in the FES.
(iii) Calculate the service demands of the FES with slow cores under different workload mixes. This step involves measurements of the response time of one request in the system while the CPU is running at its minimum frequency. Then, we have $D^{slow}(1) = R^{slow}(1)$, where $R^{slow}(1)$ is the average response time when one request is processed at a slow server.

(iv) Use the open MVA-DFS algorithm to predict average response times for the system under different workload mixes.
(v) Measure the average response times of the system under different workload mixes. This step involves measurements of the average response times with the corresponding arrival rates from the previous step.
(vi) Validate predicted results with corresponding measurements under different workload mixes.

For the first step, the queueing model is the same as that in Figure 1b. In the second step, we need to calculate the service rates of the FES with different numbers of requests in the open system. Here, we can reuse the service rate curves in Figure 3.



(a) Browsing mix (b) Shopping mix (c) Ordering mix

Fig. 6: Predicted and measured R with different mixes

The experimental results are shown in Figure 6. The results from APEM are compared with the results from measurements and the original open MVA algorithm. Note that Figures 6a, 6b and 6c have different maximum arrival rates, because of the stability condition $U_m < \alpha(\bar{N}_m)$. Since $U_m = \lambda D_m(1)$ and $\alpha(\bar{N}_m) = D_m(1)/D_m(\bar{N})$, we rewrite the stability condition as $\lambda < 1/D_m(\bar{N})$. In other words, the arrival rate cannot exceed the maximum service rate $\mu_m(\bar{N})$ in the service rate curve to make sure our algorithm is applicable. In our tests, $\mu_m(\bar{N})$ for the three mixes is 328.961, 286.660 and 600.908 per second, respectively.

As can be seen from Figures 6a, 6b and 6c, the MVA-DFS algorithm consistently performs better than the original MVA algorithm, suggesting that our service demand approximation based on DFS is a reasonable approach for such systems.

3.3. *Experiments for Semi-open Systems*

We have seen closed and open queueing systems in Sections 3.1 and 3.2, respectively. While these are the most common systems for workload generators and benchmarks, a more appropriate model may be to combine open and closed behaviours. Such hybrid systems are called semi-open systems. In semi-open systems, external users arrive following a Poisson process with rate λ. Once a user enters the system, it triggers a sequence of requests to the system before leaving. Each user will spend a think time with mean Z between receiving a response from the system and sending the next request.

We again extend the TPC-W kit. The main thread generates new EBs following an exponentially distributed time interval with a user configured rate λ. Once an EB starts, it sends the first request immediately, and then waits for an exponentially distributed think time with a user configured mean Z. After that, the EB repeats the sending and waiting procedure until it finishes (when it revisits the *Home* page again) and leaves. The expected numbers of requests per user T for the three mixes are calculated according to the transition probabilities in the TPC-W standard specification. The source code of this modified TPC-W kit can also be acquired from our website.[24]

Here, both the average response time and system throughput are performance metrics of interest. This is in contrast with open and closed networks, where the average response time is the only metric of interest. The relatively large mean think time makes it easy to estimate throughput for closed networks, and for open networks the throughput is simply the arrival rate. We set smaller mean think times in this case study because of the numerical overflow issue in the open MVA algorithm - we will discuss this in detail later. The average think time here is 0.07 seconds. The arrival rate of user sessions varies from 1 per second to 40 per second.

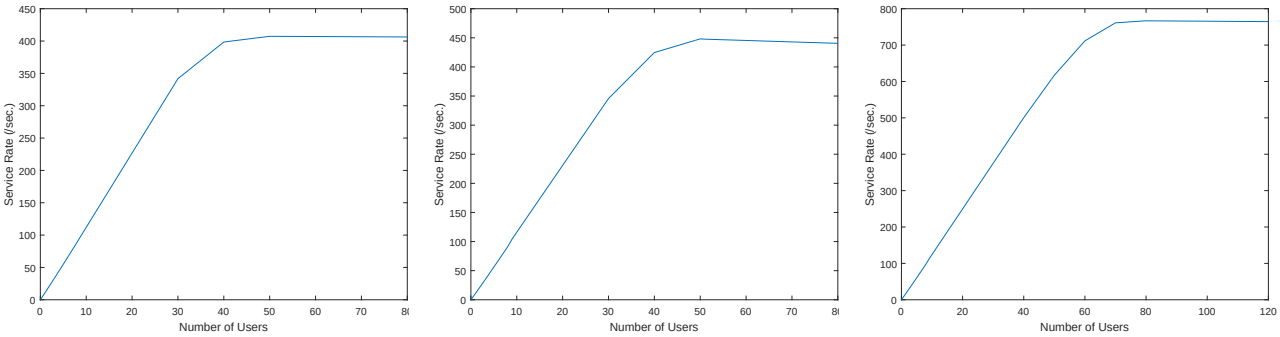
We apply APEM in the following manner:

(i) Model the system as a two-level queueing network with an FES. The upper level is an open queueing network, and the lower level is a closed queueing network.
(ii) Calculate the service demands of the FES in the lower level under different workload mixes. This step involves measurements of the throughputs (service rates) with various numbers of users in the FES.
(iii) Calculate the service demands of the FES with slow cores in the lower level under different workload mixes. This step involves measurements of the response time of one request in the system while the CPU is running at its minimum frequency. Then, we have $D^{slow}(1) = R^{slow}(1)$, where $R^{slow}(1)$ is the average response time when one request is processed at a slow server.
(iv) Use the CMVA-DFS algorithm to calculate throughputs with n ($1 \leq n \leq N$) jobs and mean think time Z. These throughputs correspond to the service rates of the FES (at request level) with n jobs in the upper level.
(v) Use the open MVA algorithm with load-dependent servers to predict average

response times and system throughputs for the system under different workload mixes. Because the arrival rate is at user session level, we need to calculate the expected number of requests per user session T for the three mixes, and then transfer all request level parameters to user session level.

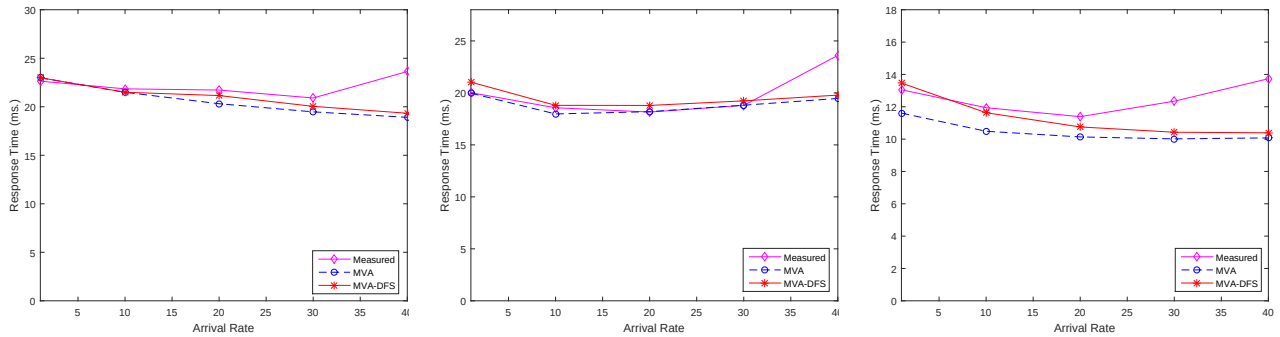
(vi) Measure the average response times and the system throughput of the system under different workload mixes. This step involves measurements of the average response times and the system throughputs with the corresponding arrival rates in the previous step.

(vii) Validate predicted results with corresponding measurements under different workload mixes.



(a) Browsing mix (b) Shopping mix (c) Ordering mix

Fig. 7: Service rate of FES with different mixes (Z=0.07 seconds)



(a) Browsing mix (b) Shopping mix (c) Ordering mix

Fig. 8: Predicted and measured R with different mixes (Z=0.07 seconds)

The service rate curves with $Z = 0$ are the same as those shown in Figure 3. The service rate curves with $Z = 0.07$ seconds are shown in Figure 7. We discuss the approach that we use to calculate T for the three mixes in Step 5 in Appendix B.

Table 2: Throughputs (in transactions per second) with browsing mix (Z=0.07 seconds)

λ	Measured	MVA	MVA-DFS
1	3.478	3.445	3.445
10	34.571	34.450	34.450
20	69.271	68.900	68.900
30	104.387	103.350	103.350
40	138.963	137.800	137.800

Table 3: Throughputs (in transactions per second) with shopping mix (Z=0.07 seconds)

λ	Measured	MVA	MVA-DFS
1	5.995	6.264	6.264
10	62.697	62.640	62.640
20	122.807	125.280	125.280
30	186.058	187.920	187.920
40	248.296	250.560	250.560

Table 4: Throughputs (in transactions per second) with ordering mix (Z=0.07 seconds)

λ	Measured	MVA	MVA-DFS
1	11.137	10.957	10.957
10	109.263	109.570	109.570
20	221.771	219.140	219.140
30	332.333	328.710	328.710
40	440.356	438.280	438.280

The average response times of the three mixes are shown in Figure 8. The measured results are compared with the results from two approaches, both of which

apply the same two-level FES technique. The first approach employs the MVA algorithms without DFS adjustment (the original CMVA algorithm and Algorithm 3), and the second one employs the MVA algorithms with DFS adjustment (Algorithm 1 and Algorithm 3). As can be seen in the figures, the MVA algorithm with the service demand estimate based on DFS is more accurate than the original MVA algorithm. In addition, both of the MVA algorithms offer relatively good estimates of throughputs, which can be seen in Tables 2, 3 and 4. As a result, we can conclude that our two-level performance model (with associated MVA algorithms) is a reasonable approach for analyzing such systems.

The two-level performance model has one limitation, which may affect the decision as to whether to use it or not. In our tests, we observe that the service rate curve converges much slower when the think time increases. As a result, the calculation of $P_m(0)$ in the open MVA algorithm encounters numerical issues (see Algorithm 3). The product involving $\alpha_m(n)$ becomes very large and can cause overflow. This is the reason why we have relatively smaller think times (to make the service rate curve converge faster) in this case study compared with previous case studies. In Section 4, we will show more tests with a larger mean think time (which is the largest Z that we can set without any overflow issues in our testbed) to demonstrate the limits of our performance model in this case study.

Because service demand ratios (instead of the service demands themselves) are used in open MVA algorithms, typical scaling techniques are inapplicable. One candidate for mitigating the overflow is to increase the range of integers and floating point numbers. We can apply a library for arbitrary precision arithmetic. One of our choices is the GNU Multiple Precision (GMP) arithmetic library.[25] GMP has been bound to many popular languages, such as C, C++, and Python. However, GMP is still limited by available memory. Another possible solution is to develop a numerically stable approximate MVA algorithm for open networks. To the best of our knowledge, there is no such approximation available in the literature. These possibilities are left for future study.

3.4. *Discussion*

APEM has several limitations. First, APEM relies on the service rate curve. As a result, any systems which are not operational (to obtain measurements) are not suitable for our method. Second, the service rate curve is an approximation based on a limited number of measured points. We suggest that the number of points should stay at a reasonable level to at least represent the shape of the curve, for example, choosing some key points on the curve. For instance, our testbed has eight cores, so we assume that the service rate with eight users in the system is the peak point on the curve. Then we can choose some key points (e.g., 1, 2, 4, 6, 8, 10, 12...) to verify this assumption, and generate the curve. Last but not least, we adopt an approximation to take DFS into account in the proposed CMVA algorithm, which may not work as well in cases when the assumptions discussed in Section 2.2 are

violated.

4. Closed versus Open versus Semi-open

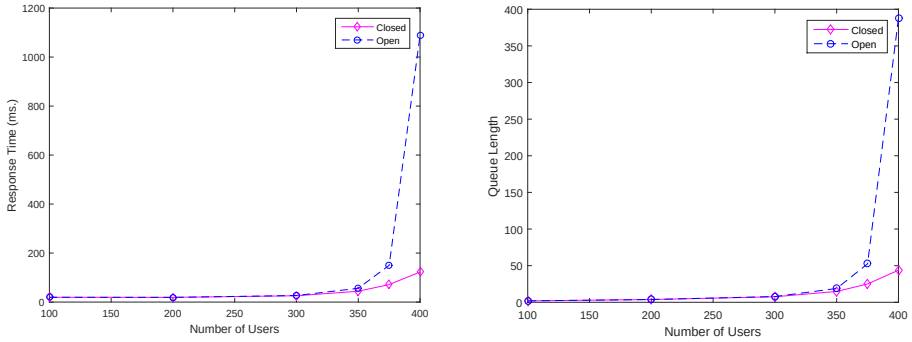
It is very important to build a queueing network which can accurately represent the system being modelled. Schroeder et al. have shown that closed and open systems can yield very different results even under the same load and with the same service demands.[26] Thus, an inappropriate choice of model may result in significant errors in performance evaluation.

4.1. *Closed versus Open*

In closed queueing systems, the number of jobs in the system is fixed. In open queueing systems, the number of requests in the system is not bounded. As a result, the number of requests could increase dramatically for large arrival rates. This may in turn cause inordinate contention among resources, and affect the performance. Here, we demonstrate the differences between a closed and an open system with a concrete example. In this example, we would like to use MVA algorithms to determine the different behaviours of closed and open systems. Both of the systems have the same service rates and workloads. The test proceeds as follows:

(i) We use the service rate curve of the shopping mix from Figure 3b, and employ Algorithm 1. We fix the mean think time $Z = 1$ second. The number of users in the system ranges from 100 to 400. The outputs of interest are the throughput X, the mean response time R, and the mean queue length at the server Q.
(ii) For the next step, we use the same service rate curve (both systems have the same service rates), and use Algorithm 2 from Section 3.2. The arrival rate λ equals the throughput X obtained from the last step (both systems have the same workloads). The outputs of interest are X, R and Q.
(iii) Finally, we compare R and Q for these two systems.

The results are shown in Figure 9, where we compare the average response times and queue lengths between the two systems. We do not need to compare their throughputs because the arrival rate λ in the open model equals the system throughput X in the closed model. When the number of jobs in the closed system varies from 100 to 300, we do not observe much difference between these two systems. However, both R and Q of these systems diverge beyond $N = 350$. When $N = 350$ in the closed system, the mean response time is 44.169 milliseconds, and the mean response time of the open system is 56.214 milliseconds. The difference is 27.27%. We have similar observations for the values of Q. The extreme case happens when $N = 400$ in the closed system. The mean response time of the open system is about nine times larger than for the closed system, and so is the mean queue length. In summary, closed and open systems may have significant differences under heavy loads. This conclusion is consistent with the observations in Schroeder et al.'s work.[26]



(a) Response time comparison (b) Queue length comparison

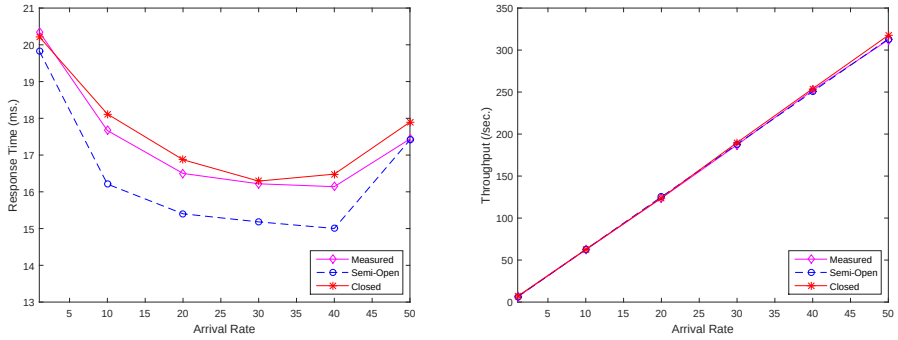
Fig. 9: Comparison between closed and open systems

4.2. *Closed versus Semi-open*

Both closed and open systems are well studied in the literature. However, semi-open models are not as well understood. Considering an e-commerce website, neither the assumption of a closed system nor the assumption of an open system is realistic. On the one hand, a closed system assumes that the number of jobs in the system is fixed. On the other hand, an open system assumes that requests are sent to the server without any dependence on previous ones. While closed and open networks can be used to model particular scenarios (e.g., stress tests), semi-open systems can model realistic job structures and user behaviours of e-commerce systems. In an e-commerce system, customers arrive at the website with an arrival rate. Once a customer arrives, they start browsing the website. After a click on a webpage, the customer may think for a period of time, and then make another click. Each click on the webpage sends an HTTP request to the system. A customer repeats the click and think behaviours until they leave the website. Such customer behaviours are captured by a semi-open model.

While semi-open models may better reflect reality, they are more difficult to analyze. A hierarchical flow-equivalent aggregation technique may be deployed for such cases. Although flow-equivalent aggregation has been shown to be accurate in the literature, hierarchical modelling brings more challenges. The performance model in our case study in Section 3.3 only has a two-level FES, but new numerical issues arose that did not appear for open or closed models.

It would be nice if we could use a closed or an open model to evaluate the performance of a semi-open system. Suppose that we know the average number of users in the system, then we can build a closed model for the system. Let us extend the results in the semi-open system in Section 3.1 with a larger mean think time, $Z = 0.7$ seconds. According to system logs, we calculate the average number of users N (rounded up to the closest integer), and then use Algorithm 1 to compute



(a) Response time comparison (b) Throughput comparison

Fig. 10: Comparison between semi-open and closed systems

R and X for the closed model. For example, the average number of users is five in the first case (when $\lambda = 1$). Then we use $N = 5$ as our input for the closed MVA algorithm, and obtain $R = 20.218$ milliseconds and $X = 6.942$ per second. As can be seen from Figure 10, the MVA algorithm for closed systems produces very accurate results compared to measured values in all cases. In Figure 10, we compare the average response times and throughputs with $\lambda = 1$, 10, 20, 30, 40 and 50. We have $N = 5$, 45, 89, 136, 182 and 228, respectively.

One of the conditions for this approach is that the average number of users is measurable. In addition, we do not observe a large variance in the number of users in our tests. The accuracy of this approximation in cases where this variance is larger needs to be verified in future work.

5. Multi-class Models

For completeness, we extend the two proposed algorithms - Algorithm 1 and Algorithm 2 - to the cases of multi-class closed and open networks, respectively. Algorithm 3 in multi-class semi-open models can be found in Chapter 14 of Menascé et al.'s work.[2]

For closed networks, consider that there are C classes of transactions, where the job population vector is given by $\vec{N} = (n_1, n_2, \ldots, n_C)$, and each class has a mean think time Z_c $(1 \leq c \leq C)$. The probability that a job of class c arrives to a slow server at the mth queue with n_c jobs is given by:

$$P_{m,c}^{slow}(n_c) = Exp(-\frac{sn_c}{Z_c K_m}). \tag{3}$$

Using the results from Eq. (3), the formula for class-level service demands with DFS adjustment at the mth queue is given by:

$$D_{m,c}(n_c) = D_{m,c}^{unadjusted}(n_c)\big(1 - P_{m,c}^{slow}(n_c)\big) + D_{m,c}^{slow}(n_c)P_{m,c}^{slow}(n_c). \tag{4}$$

Then, the multi-class CMVA-DFS algorithm iterates over all of the classes to compute the mean response times, the throughputs, and the mean queue lengths, respectively. Similar to Algorithm 1, we first introduce the formula for the class-level mean response times:

$$R_{m,c}(\vec{N},t) = D_{m,c}(\vec{N},t)\big(1 + Q_m(\vec{N} - 1_c, t+1)\big).$$

Here, $\vec{N} - 1_c$ is the job population vector with one less class c job in the system. The system throughput of class c is calculated by

$$X_c(\vec{N}) = n_c/\big(Z_c + \sum_{m=1}^{M} R_{m,c}(\vec{N},t)\big),$$

where $0 \leq n_c \leq N_c$. The mean queue length at the mth queue is

$$Q_m(\vec{N},t) = \sum_{c=1}^{C} X_c(\vec{N}) R_{m,c}(\vec{N},t).$$

For open networks, we also consider that there are C classes of transactions, and each class has arrival rate λ_c $(1 \leq c \leq C)$. We rewrite the slow server probability in Algorithm 2 for multi-class models as follows:

$$P_{m,c}^{slow}(n_c) = Exp(-\frac{sn_c\lambda_c}{K_m}).$$

The formula for class-level service demands in open networks is exactly the same as Eq. (4). As mentioned in Chapter 14 of Menascé et al.'s work,[2] we assume that the service demand ratio α of any load-dependent node is class independent, which means that $\alpha_{m,c}(n) = \alpha_m(n)$ for all c. Since the utilization at the mth queue is also load independent, the formulas for $P_m(0)$ and Q_m in Algorithm 2 also do not change in the multi-class open model. To determine the class-level mean queue lengths and mean response times, we introduce two new equations as follows:

$$Q_{m,c} = Q_m(U_{m,c}/U_m),$$

where $U_{m,c} = \lambda_c D_{m,c}(1)$, and

$$R_{m,c} = Q_{m,c}/\lambda_c.$$

6. Related Work

Workload characterization and parameterization is one of the most challenging aspects in employing a multi-class MVA algorithm. Linear regression is one of the most widely used approaches to estimate resource demands. Zhang et al. applied a regression-based approximation of the CPU demands of online web transactions.[16,17] To minimize the absolute error, they adopted the non-negative LSR provided by MATLAB to obtain resource demands. They then used the approximation results in a multi-tier queueing model. Kraft et al. also applied linear regression and a maximum likelihood technique to parameterize a performance

model for an industrial ERP system.[18] However, regression techniques suffer from the well-studied problem of multicollinearity (see Chapter 15 in Jain's work),[27] which can lead to unreliable predictions for demands and very wide confidence intervals for predicted demands. Mi et al. and Kalbasi et al. have shown that the accuracy of regression-based techniques critically depends on the quality of monitoring data used in the regression analysis.[15,28]

Zhang et al. showed that it is not true that the service demand is load independent for modern processors with DFS and Hyper-Threading Technology (HTT) features.[19] Their study demonstrated that the CPU demand can be modelled as a polynomial function of the CPU utilization. The polynomial coefficients are inferred from measured CPU utilizations and response times. They built their experimental environment based on a server with a multi-core processor, and then compared the accuracy of their proposed estimation method with a load-independent regression method and a load-dependent estimation method proposed by Kumar et al.[29] Using various workloads, they showed that their estimation method is more accurate. It is an interesting approach to explore the load-dependent behaviours in a multi-core processor. However, it requires a very large number of measurements to parameterize the polynomial function, because the estimate is based on measurements of not only the CPU utilization but also response time. In addition, they did not adjust the CPU demand to take into account the effects of DFS.

Different MVA approaches have been proposed for open, closed, and semi-open queueing networks with load-dependent queues. However, to the best of our knowledge, the literature is lacking awareness of the impact of DFS on performance models. For closed systems, Seidmann's approximation is widely used to address MVA's numerical instability issues.[30,31,32] However, it is only applicable for multi-server queues in which all servers are load independent. For closed systems with generic load-dependent queues, Casale introduced the CMVA algorithm to overcome the numerical instability issue.[10] For open systems with load-dependent queues, a multi-class MVA algorithm was introduced by Menascé et al. (Chapter 14).[2] For semi-open systems with load-dependent queues, the common analytic procedure is to treat the open and closed classes separately. Bruell et al. first presented the MVA algorithm for load-dependent semi-open networks,[33] details can also be found in Chapter 8 of Bolch et al.'s work.[34] None of these MVA algorithms take into account the effect of DFS on multi-core computer systems.

7. Conclusions

To address performance modelling for a multi-core computer system which cannot be modelled as a product-form queueing network, we proposed to solve a flow-equivalent aggregated model using MVA algorithms, where service demands are adjusted to compensate for the effects of DFS. In order to illustrate the applicability of our performance model, we extended the original TPC-W benchmark to open and semi-open systems, then we applied the APEM method in three case studies.

The performance metrics (average response times) are validated with the experimental results. We have shown that our MVA algorithms with DFS adjustment outperform the original MVA algorithms in all three case studies. We also discussed and compared closed, open, and semi-open models and their limitations. Last but not least, we extended the proposed MVA algorithms in APEM to multi-class models.

There are several potential directions for future work. Firstly, we would like to improve the time and space complexities of the CMVA algorithm. Secondly, we would like to develop numerically stable MVA algorithms without overflow/ underflow for open and semi-open systems.

Acknowledgment

The work reported in this paper was supported by the Ontario Research Fund and the Natural Sciences and Engineering Research Council of Canada.

Appendices

Appendix A. Outlier Detection

In this paper, we use the outlier detection method presented by Ben-Gal.[35] The method takes three input parameters - the average μ, the variance σ^2, and the confidence level $1-\alpha$. The outlier region is defined by

$$out(\alpha, \mu, \sigma^2) = \{x : |x - \mu| > z_{1-\alpha/2}\sigma\},$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$-quantile of a unit normal variate (see Chapter 13 in Jain's work).[27] In our tests, we choose $\alpha = 0.1$ (90% confidence interval), which is a typical value.

Appendix B. Calculation of T for Semi-open Systems

We assume that every user session starts from a *Home* page access, and ends with the next *Home* page access after several other page accesses. A user-web interaction transfers from one page to another with a fixed probability. The transition probabilities between pages in the three mixes are shown in Tables 5, 6 and 7. The abbreviations are listed in Table 8. A blank entry in a table means zero (no transition is possible). We would like to calculate the expected number of requests between H1 and H2.

We use the following equation to calculate the expected number of requests per user:

$$T_i = 1 + \sum_j p_{i,j} T_j,$$

Table 5: Transition probabilities for browsing mix

	AC	AR	BS	BC	BR	CR	H1	H2	NP	OD	OI	PD	SRq	SRs	SC
AC								0.9878					0.0122		
AR	0.9000							0.1000							
BS								0.4607				0.0652	0.4683		0.0057
BC								0.0342					0.9658		
BR				0.9200				0.0396							0.0404
CR					0.9146			0.0474					0.0380		
H1			0.3792						0.3793		0.0103		0.1871		0.0440
H2								1							
NP								0.0299				0.9569	0.0074		0.0058
OD								0.0802					0.9198		
OI								0.0523		0.8334			0.1143		
PD		0.0047						0.8300				0.1403	0.0141		0.0109
SRq								0.0788						0.9168	0.0044
SRs								0.3674				0.6195	0.0074		0.0057
SC						0.4099		0.4784							0.1116

Table 6: Transition probabilities for shopping mix

	AC	AR	BS	BC	BR	CR	H1	H2	NP	OD	OI	PD	SRq	SRs	SC
AC								0.9952					0.0047		
AR	0.9000							0.1000							
BS								0.0167				0.0305	0.9456		0.0072
BC								0.0084					0.9916		
BR				0.4614				0.1932							0.3453
CR					0.8667			0.0094					0.1239		
H1			0.3124						0.3125		0.0469		0.0308		0.2973
H2								1							
NP								0.0156				0.9580	0.0049		0.0215
OD								0.0069					0.9931		
OI								0.0072		0.8801			0.1127		
PD		0.0058						0.0774				0.0456	0.7316		0.1396
SRq								0.0635						0.8501	0.0864
SRs								0.2657				0.6638	0.0010		0.0695
SC						0.2585		0.6968							0.0447

Table 7: Transition probabilities for ordering mix

	AC	AR	BS	BC	BR	CR	H1	H2	NP	OD	OI	PD	SRq	SRs	SC
AC								0.8349					0.1651		
AR	0.9000							0.1000							
BS								0.0001				0.0332	0.9666		0.0001
BC								0.0002					0.9998		
BR				0.8000				0.1454							0.0546
CR					0.9900			0.0002					0.0098		
H1			0.0499						0.0500		0.0270		0.0026		0.8705
H2								1							
NP								0.0504				0.9439	0.0034		0.0023
OD								0.9940					0.0060		
OI								0.1168		0.8801			0.0031		
PD		0.0099						0.3651				0.1871	0.0720		0.3658
SRq								0.0815						0.9001	0.0184
SRs								0.0486				0.7332	0.2181		0.0001
SC						0.9500		0.0419							0.0081

where $p_{i,j}$ is the probability that a user visit transfers from page i to page j (corresponds to entries in Tables 5, 6 and 7), T_i and T_j are the expected numbers of steps from page i and page j to the final state, respectively. In Table 5, we have

Table 8: TPC-W page name abbreviations

AC	Admin Confirm
AR	Admin Request
BS	Best Sellers
BC	Buy Confirm
BR	Buy Request
CR	Customer Regist.
H1	Home (Start)
H2	Home (End)
NP	New Products
OD	Order Display
OI	Order Inquiry
PD	Product Detail
SRq	Search Request
SRs	Search Results
SC	Shopping Cart

$T_{H1} = 1+0.3792T_{BS}+0.3797T_{NP}+0.0103T_{OI}+0.1871T_{SR_q}+0.0440T_{SC}$. Similarly, we can write equations for T_{BS}, T_{NP}, T_{OI}, T_{RS_q}, T_{SC}, and so on. We continue until we have all of the equations (14 equations in our case) which calculate T_i from page i to H2, and solve those equations. Finally, the expected numbers of requests per user for the browsing, shopping and ordering mixes are 3.445, 6.264 and 10.957, respectively.

References

1. M. Reiser and S. S. Lavenberg, Mean-value analysis of closed multichain queuing networks, *Journal of the ACM (JACM)* **27**(2) (1980) 313–322.
2. D. A. Menascé, V. A. Almeida and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example* (Prentice Hall Professional, 2004).
3. V. Pallipadi and A. Starikovskiy, The ondemand governor, in *Proceedings of the Linux Symposium*, **2**, (Linux Symposium, 2006), pp. 215–230.
4. D. A. Menascé, Modeling the tradeoffs between system performance and cpu power consumption, in *Proceedings of the 2015 Computer Measurement Group Conference*, (CMG, 2015).
5. G. Dhiman and T. S. Rosing, Dynamic voltage frequency scaling for multi-tasking systems using online learning, in *Proceedings of the 2007 international symposium on Low power electronics and design*, (ACM, 2007), pp. 207–212.
6. A. Wierman, L. L. Andrew and A. Tang, Power-aware speed scaling in processor sharing systems, in *Proceedings of the 28th Conference on Computer Communications*, (IEEE, 2009), pp. 2007–2015.
7. A. Wierman, L. L. Andrew and A. Tang, Power-aware speed scaling in processor sharing systems: Optimality and robustness, *Performance Evaluation* **69**(12) (2012) 601–622.

8. L. Zhang and D. G. Down, Approximate mean value analysis for multi-core systems, in *Proceedings of the 2015 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, (SCS, 2015), pp. 1–8.
9. K. M. Chandy, U. Herzog and L. Woo, Parametric analysis of queuing networks, *IBM Journal of Research and Development* **19**(1) (1975) 36–42.
10. G. Casale, A note on stable flow-equivalent aggregation in closed networks, *Queueing Systems* **60**(3-4) (2008) 193–202.
11. E. D. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis using Queueing Network Models* (Prentice-Hall, Inc., 1984).
12. Transaction Processing Performance Council, TPC-W official website http://www.tpc.org/tpcw, (2005), last accessed 24 Feburary 2016.
13. G. Casale, N. Mi, L. Cherkasova and E. Smirni, Dealing with burstiness in multi-tier applications: Models and their parameterization, *IEEE Transactions on Software Engineering* **38**(5) (2012) 1040–1053.
14. N. Mi, G. Casale, L. Cherkasova and E. Smirni, Burstiness in multi-tier applications: Symptoms, causes, and new models, in *Proceedings of 9th ACM/IFIP/USENIX International Conference on Middleware*, (Springer-Verlag New York, Inc., 2008), pp. 265–286.
15. N. Mi, L. Cherkasova, K. Ozonat, J. Symons and E. Smirni, Analysis of application performance and its change via representative application signatures, in *Proceedings of Network Operations and Management Symposium*, (IEEE, 2008), pp. 216–223.
16. Q. Zhang, L. Cherkasova, N. Mi and E. Smirni, A regression-based analytic model for capacity planning of multi-tier applications, *Cluster Computing* **11**(3) (2008) 197–211.
17. Q. Zhang, L. Cherkasova and E. Smirni, A regression-based analytic model for dynamic resource provisioning of multi-tier applications, in *Proceedings of 4th International Conference on Autonomic Computing*, (IEEE, 2007), pp. 27–36.
18. S. Kraft, S. Pacheco-Sanchez, G. Casale and S. Dawson, Estimating service resource consumption from response time measurements, in *Proceedings of 4th International ICST Conference on Performance Evaluation Methodologies and Tools*, (ICST, 2009).
19. Z. Zhang, S. Li and J. Zhou, Estimate load-dependent service demand for modern CPU, *Information Technology Journal* **12** (2013) 632–639.
20. JBoss Group, JBoss application server http://jbossas.jboss.org, (2005), last accessed 24 Feburary 2016.
21. MySQL AB, MySQL http://www.mysql.com, (2008), last accessed 24 Feburary 2016.
22. T. Horvath, TPC-W J2EE implementation http://www.cs.virginia.edu, (2008), last accessed 24 Feburary 2016.
23. L. Zhang, *Performance Models for Legacy System Migration and Multi-core Computers - an MVA Approach*, PhD thesis, McMaster University, (Ontario, Canada, 2015).
24. L. Zhang, TPC-W kits for open and semi-open systems http://www.cas.mcmaster.ca, (2015), last accessed 24 Feburary 2016.
25. T. Granlund *et al.*, GMP, the GNU multiple precision arithmetic library https://gmplib.org, (2014), last accessed 24 Feburary 2016.
26. B. Schroeder, A. Wierman and M. Harchol-Balter, Open versus closed: A cautionary tale, in *Proceedings of 3rd Symposium on Networked Systems Design and Implementation*, **3**, (USENIX Association, 2006), pp. 239–252.
27. R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling* (John-Wiley, 1991).
28. A. Kalbasi, D. Krishnamurthy, J. Rolia and S. Dawson, DEC: Service demand estimation with confidence, *IEEE Transactions on Software Engineering* **38**(3) (2012)

561–578.
29. D. Kumar, L. Zhang and A. Tantawi, Enhanced inferencing: Estimation of a workload dependent performance model, in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, (ICST, 2009).
30. A. Seidmann, J. Paul and S. Shalev-Oren, Computerized closed queueing network models of flexible manufacturing systems: A comparative evaluation, *Large Scale Systems* **12** (1987) 91–107.
31. X. Liu, J. Heo and L. Sha, Modeling 3-tiered web applications, in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, (IEEE, 2005), pp. 307–310.
32. Y. Chen, S. Iyer, X. Liu, D. Milojicic and A. Sahai, Translating service level objectives to lower level policies for multi-tier services, *Cluster Computing* **11**(3) (2008) 299–311.
33. S. C. Bruell, G. Balbo and P. Afshari, Mean value analysis of mixed, multiple class bcmp networks with load dependent service stations, *Performance Evaluation* **4**(4) (1984) 241–260.
34. 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, 1998).
35. I. Ben-Gal, Outlier detection, in *Data Mining and Knowledge Discovery Handbook*, eds. O. Maimon and L. Rokach (Springer, 2005) pp. 131–146.