Approximate Mean Value Analysis for Multi-core Systems

Lei Zhang
Department of Computing and Software
McMaster University
1280 Main Street West
Hamilton, ON L8S 4K1
Canada
Email: zhangl64@mcmaster.ca

Douglas G. Down
Department of Computing and Software
McMaster University
1280 Main Street West
Hamilton, ON L8S 4K1
Canada
Email: downd@mcmaster.ca

***Abstract*—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 modeling, it fails to address several features of multi-core platforms. 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 also achieves accuracy for performance prediction for multi-core platforms. Our contributions are: We present a flow-equivalent performance model designed specifically to address multi-core computer systems. We identify the influence on the CPU demand of the effects of Dynamic Frequency Scaling (DFS) and Hyper-Threading Technology (HTT). We adopt an approximation technique to estimate resource demands to parameterize the MVA algorithm. We use a modified Conditional MVA (CMVA) algorithm to address the potential numerical instability. To validate the application of our method, we investigate a case study of an e-commerce web server which is equipped with diverse classes of user requests. We show that our method achieves better accuracy compared with other commonly used MVA algorithms.**

***Keywords*—performance evaluation, performance models, mean value analysis, flow-equivalent aggregation, service demand estimation.**

I. Introduction

The exact Mean Value Analysis algorithm for the analysis of closed queueing networks with product-form steady-state distribution was first published by Lavenberg and Reiser in 1980 [21]. 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 terms node and resource are used interchangeably in this paper).

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, *e.g.*, simultaneous resource possession, locking behaviors in database servers, priority scheduling, high service demand variability, and process synchronization (see Chapter 15 in [16]). As the steady-state distribution is not product-form, the MVA algorithm is infeasible in such cases.

Secondly, it is often not straightforward to determine the required parameters for the MVA algorithm. The Service Demand Law, in which the resource demand is equal to the resource utilization divided by the throughput, is one of the most popular approaches (see Chapter 3 in [16]). 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 [3], [17], [18], [24], [25]. 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 [10], [26].

Thirdly, multi-core processors cause difficulties in estimating the CPU demand. One challenge comes from systems employing Dynamic Frequency Scaling (DFS) techniques [12]. With DFS, a CPU/processor can lower its frequency when lightly loaded, and increase its frequency when highly loaded. In other words, the CPU demand can be adjusted dynamically according to the workload. Another challenge that we consider is Intel's Hyper-Threading Technology (HTT) [14]. With HTT, one core can run multiple instructions simultaneously to exploit Instruction Level Parallelism (ILP). With these two features, the offered CPU demand depends in a nontrivial way on the load in the system. Finally, when employing load-dependent closed queueing networks to model systems with multi-core processors, the load-dependent MVA algorithm invariably becomes unstable [2]. It could go so far as to produce negative values of system throughput or response time.

To address these problems, we introduce a new method named APEM, Approximate Performance Evaluation method for Multi-core computer systems. In APEM, we propose to use a performance model based on the flow-equivalent aggregation method, which is often employed to solve non-product-form queueing network models. The main goal of APEM is to build a robust performance model with a limited number of

TABLE I: Notation

M	number of resources (devices)
N	number of users
Z	average think time
D_m	resource demand at the mth resource (*e.g.*, CPU and disk)
K_m	number of servers at the mth resource
R	average response time
X	throughput
Q	average queue length

measurements on the system. To do this, we present an approximate service rate curve to parameterize the performance model, addressing the effect of HTT. The impact of DFS, while also addressed by the service rate curve, requires an additional approximation to adjust the service rate (or more precisely, its inverse: service demand). Finally, we embed our approximations into a numerically stable load-dependent algorithm - Conditional Mean Value Analysis (CMVA) - to solve the queueing network model [2]. In order to validate our method, we implement a system using the TPC-W benchmark [6], and compare experimental results with those generated by our algorithm. For reference, the notation presented in Table I is used in the remainder of the paper.

II. Background

Estimating resource demands, especially CPU demand, remains an area of active research [3], [10], [17], [18], [24], [25], [26]. For load-independent performance models, average resource demands are constant. For multi-server queueing networks, the demands of resources that have multiple servers are load-dependent. However, if we assume that single server demand is a constant value, we can calculate the corresponding resource demand as follows,

$$D_{resource}(n) = \begin{cases} D_{server}/n & \text{if } n \leq K \\ D_{server}/K & \text{if } n > K, \end{cases} \quad (1)$$

where $D_{resource}(n)$ is the resource demand, D_{server} is the demand of an individual server, K is the number of servers at the node, and n is the number of jobs at the node. As can be seen from (1), $D_{resource}$ is constant when n exceeds K. However, these assumptions do not hold for modern computers with multi-cores. There are two key reasons behind this - Dynamic Frequency Scaling (DFS) and Hyper-Threading Technology (HTT).

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 [20]. 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 the value of *up_threshold*, Linux sets the CPU to run at the highest frequency. When the load falls below the same threshold, Linux sets the CPU to run at the next lowest frequency that can still keep 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

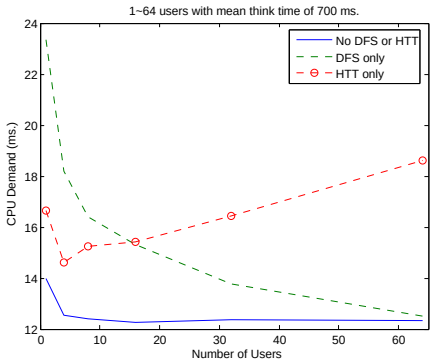


Fig. 1: DFS and HTT impact on CPU demand

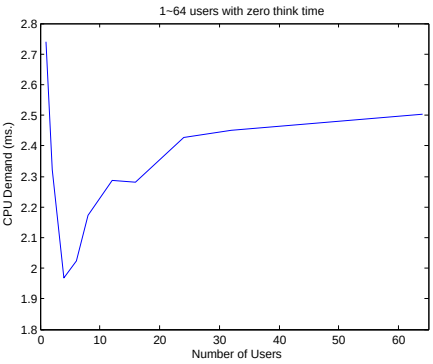


Fig. 2: CPU demand with both DFS and HTT enabled

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.

HTT is used to improve parallelization of computations performed on multi-core processors. In HTT-enabled systems, each physical core is addressed by the operating system to form two logical cores, and workload is shared between them when possible. Linux-based operating systems treat each logical core as a physical core, which causes a potential problem for performance modeling. Since a multi-core CPU has shared execution resources (*e.g.*, execution engines and caches), logical cores become slower due to the increasing contention for internal resources among multiple threads under high load [13]. In other words, the CPU demand increases when the load increases in an HTT-enabled system.

We illustrate these issues with an example that employs the TPC-W benchmark. The testbed has an Intel i7-2600 quad-core processor (more hardware and software details can be found in Section IV). The CPU demand is calculated by the Service Demand Law. The CPU utilization is monitored by the *sysstat* utility, and the throughput is obtained from transaction logs. As can be seen from Figure 1, if both DFS and HTT are disabled, the CPU demand remains constant when the load increases. Here, the system could be analyzed with load-independent techniques. If only DFS is enabled, the CPU demand decreases in a nonlinear manner with increasing load and differs by

a factor of almost two between light and heavy loads. The scaling factor corresponds to the CPU frequency ratio (3.4/1.6 GHz). If only HTT is enabled, the CPU demand gradually increases with the load in a linear fashion and almost doubles at high load (as compared to light load).

Observation 1. *If both DFS and HTT are enabled, the demand scaling has both increasing and decreasing factors and the final shape of the demand curve depends on which effect is dominant.*

Figure 2 shows the CPU demand with both DFS and HTT enabled. As can be seen, the effect of DFS dominates when the CPU is lightly loaded, while the effect of HTT becomes prominent when the CPU is highly loaded. As mentioned before, we cannot use a queueing network with load-independent servers to model such a system. In addition, the shape of the curve does not match a typical multi-server MVA algorithm (see Chapter 15 in [23]), which is parameterized using (1). As a result, a general load-dependent MVA algorithm is called for in this case (details of the algorithm can be found in Chapter 14 in [16]).

However, the load-dependent MVA algorithm has issues with numerical stability. It may exhibit numerical difficulties under heavy load conditions which eventually result in unreasonable results, such as negative throughputs, response times and queue lengths. The problem is that the probability of a resource being idle is calculated in every iteration of the load-dependent MVA algorithm. The calculation is as follows:

$$P_m(0|n) = 1 - \sum_{i=1}^{n} P_m(i|n), \tag{2}$$

where $P_m(i|n)$ is the probability that i jobs are at the mth resource when a total of n jobs are in the system. When the utilization is close to one, (2) can yield negative values due to round-off errors. Those errors propagate as the MVA algorithm iterates. Subsequently, other calculations which have direct or indirect dependence on (2) may result in negative values, such as response times, throughputs and queue lengths. Additional scaling techniques are required to prevent floating-point range exceptions, and it is not clear that such techniques can always be effective [4].

In order to address the numerical instability of the load-dependent MVA algorithm, we adopt the Conditional MVA (CMVA) algorithm, presented by Casale [2], 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. This queue length formula avoids the computations of the state probabilities at each node, and as a consequence, overcomes the limitation. The case study in our paper can be considered as a demonstration of the applicability of Casale's work. In Section IV, we will present a concrete example in which the CMVA algorithm is implemented with our service demand approximations.

III. The APEM Method

In this section, we present the APEM method - a load-dependent performance model for multi-core platforms solved by the CMVA algorithm. With the load-dependent feature of our model, we address the effects of HTT and DFS. For the effects of DFS, we must additionally adjust the service demand. Finally, we present a CMVA algorithm with service demand adjustment for DFS - the CMVA-DFS algorithm.

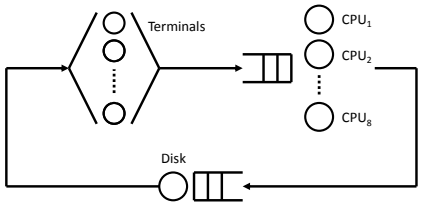


Fig. 3: Queueing network model with two resources

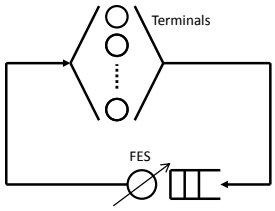


Fig. 4: Queueing network model FES

A. Performance Model

We choose an approximate solution for non-product-form queueing networks, the flow-equivalent aggregation technique. The system under study is approximated by a load-dependent resource, called a Flow-Equivalent Server (FES) (see Chapter 36 in [7]). The service rate of the FES with n jobs present is equal to the observed throughput of the system with n jobs. Finally, we can solve the equivalent network model using a product-form solution, *i.e.*, the load-dependent MVA algorithm. For instance, a multi-core computer system can be modeled as a queueing network with two resources - CPU and disk (see Figure 3) - which for us does not have a product-form solution. However, we can use the flow-equivalent aggregation approach to replace the system by a load-dependent queue (see Figure 4).

B. Service Demand Adjustment

The effect of DFS needs to be taken into account in the parameterization. Consider a scenario where a core is idle for quite a while and 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 time of the job is increased due to the scaled frequency during this interval. The parameterization of the MVA algorithm can be effectively enhanced with a corresponding service demand adjustment.

Observation 2. *There exists a probability that an upcoming job will be (partially) processed at a slow rate, because 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.*

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. We define this 10 milliseconds as the *sampling interval*. 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 an arriving job to an idle server and the last departure is larger than the sampling interval. The probability is given by $P(X > s)$, where X 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

$$P_{slow}(n) = P(X > s|n) \approx e^{-\frac{sn}{ZK}}, \quad (3)$$

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. Note that the approximation in (3) is due to the fact that this expression is an equality only if all of the users are "thinking". This approximation should work well for sufficiently large think times.

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

$$D'(i) = D(i) \times (1 - P_{slow}(n)) + D_{slow}(i) \times P_{slow}(n), \quad (4)$$

where $D(i)$ is the CPU demand with n jobs at the CPU, and $D_{slow}(i)$ is the service demand of the CPU with one core running at the minimum frequency.

Formally, we make the following assumptions on the estimate:

1) The arrivals follow an exponential distribution.
2) If an arrival enters a "slow" server, then the entire job will be processed at the slowest possible rate.
3) Each logical core works independently.
4) The average think time is sufficiently large.

Thus, 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, while others may not, because the core's frequency will be scaled up at the next sampling point. 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. Finding ways to refine the approximation is of future research interest.

C. Conditional MVA

Algorithm 1 presents the CMVA algorithm with the service demand adjustment for DFS. We call this algorithm CMVA-DFS. In the application of CMVA-DFS, we assume that all the resources are load-dependent.

Algorithm 1 The CMVA-DFS algorithm

Input:
$Z, M, N, D_m(n), D_{m,slow}(n), s, K_m$
Output:
Q, X, R
Initialization:
if m is a DFS-enabled resource **then**
for $n = 1 \to N$ **do**
$P_{slow}(n) = Exp(-s \times n/(Z \times K_m))$
$D_m(n) = D_m(n) \times (1 - P_{slow}(n)) + D_{m,slow}(n) \times P_{slow}(n)$
end for
end if
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 $D_m(n,t) = D_m(n-1,t) \times X(n-1,t)/X(n-1,t+1)$
end if
end for
for $m = 1 \to M$ **do**
$R_m(n,t) = D_m(n,t) \times (1 + Q_m(n-1,t+1))$
end for
$X(n,t) = n/(Z + \sum_{m=1}^{M} R_m(n,t))$
for $m = 1 \to M$ **do**
$Q_m(n,t) = D_m(n,t) \times X(n,t) \times (1 + Q_m(n-1,t+1))$
end for
end for
end for

IV. TPC-W Case Study

The TPC-W benchmark models an online bookstore [6]. It assesses the performance of Online Transaction Processing (OLTP) systems. Every client of the system is an Emulated Browser (EB) that issues HTTP requests (in total 14 different web interactions) to the web server. Between two consecutive web interactions, there is a "thinktime" which is defined by an exponential distribution. TPC-W defines three different workload mixes - browsing, shopping, and ordering - to simulate different user behaviors.

We built our testbed on a Dell desktop computer equipped with an Intel i7-2600 quad-core processor (octo-core logically with HTT), 8GB RAM, and a 1TB disk (7200 RPM). For the operating system, we use 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 the clients' requests and issues queries to the database in order to retrieve the requested data. For the web server and the database server, we use JBoss 3.2.7 [8] and MySQL 5.1 [19], respectively.

We choose average response time as our performance metric of interest. The number of EBs varies from 10 to 1000, and we set the average "thinktime" (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.

TABLE II: Load-dependent MVA numerical instability example

Shopping mix with Z = 3.5 sec.		
N	500	600
X (/sec.)	142.168	-107.969
R (sec.)	0.017	-9.057

We apply APEM in the following manner:

1) Model the non-product-form system as a closed queueing network with an FES (as seen in Figure 4).
2) Calculate the service demands of the FES under different workload mixes. This step involves measurements of the throughputs (service rates) with various number of users in the FES.
3) Calculate the service demands of the FES with slow cores under different workload mixes. This step involves measurements of the throughputs with one user in the FES while the CPU is running at its minimal frequency.
4) Use the CMVA-DFS algorithm to predict average response times for the system under different workload mixes.
5) 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 in the previous step.
6) Validate predicted results with corresponding measurements under different workload mixes.
7) Predict system performance with desired parameters under different workload 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 numbers 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 5. 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, 4, 6, 8 and 16 in Figure 5. 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. For the third step, we calculate the adjusted service demands using (4). The service rate of cores running at the minimum frequency can be measured by fixing the CPU frequency at its minimum, or the arrival rate at a small value (to ensure the CPU has sufficient idle time to scale down its frequency to the minimum). These curves are also shown in Figure 5.

Before presenting results from the CMVA-DFS algorithm, we would like to illustrate the numerical instability of the generic load-dependent MVA algorithm by a concrete example. The MVA algorithm is parameterized using the service rate curve in Figure 5b. As can be seen in Table II, the load-dependent MVA algorithm works fine when $N = 500$, but produces negative results (for X and R) when $N = 600$. As mentioned in Section II, this example illustrates the algorithm yielding unreasonable outputs due to iterative error propagation.

TABLE III: Outliers in browsing mix ($Z = 3.5$ sec.)

$N = 1000$			
class	average (ms.)	std (ms.)	outlier (ms.)
admin	20.019	102.341	$x > 188.370$
search	24.503	116.907	$x > 216.815$

TABLE IV: Outliers in browsing mix ($Z = 3.5$ sec.)

N	R (ms.)	total trans.	outlier%	R without outliers (ms.)
500	27.593	255940	1.65%	20.663
1000	47.028	506543	2.70%	23.224

Finally, the experimental results are shown in Figure 6. The results from APEM are compared with the results from measurements, the multi-server MVA algorithm parameterized by the Service Demand Law, and the original CMVA algorithm. A Java implementation of the multi-server MVA algorithm used here can be found in Chapter 15 in [23].

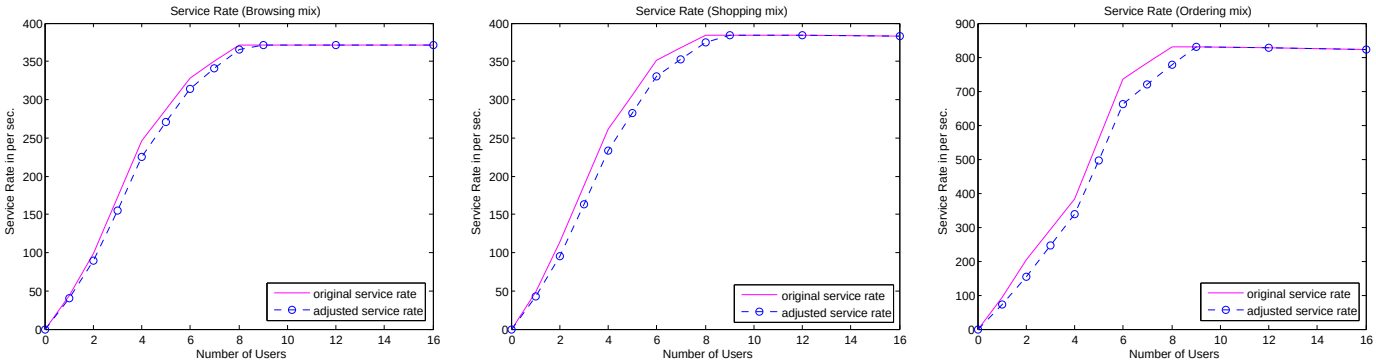
A small number of abnormally large response times are observed during our experiments - these values have a significant effect on the resulting average. To try to determine the cause of these very slow response times, we have monitored cache miss rates, memory utilizations, locking times, and also profiled the database server. We finally determined the root cause to be JBoss logging. One simple solution is to disable the logging. Another solution is to apply outlier detection techniques on our data, because one may prefer to keep the JBoss logging for administrative purpose. We decide to use the latter in this paper. The former produces similar results, they are omitted due to space considerations.

We choose a univariate method for outlier detection as presented by Ben-Gal in [1]. 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

$$out(\alpha, \mu, \sigma^2) = \{x : |x - \mu| > z_{1-\alpha/2}\sigma\}, \qquad (5)$$

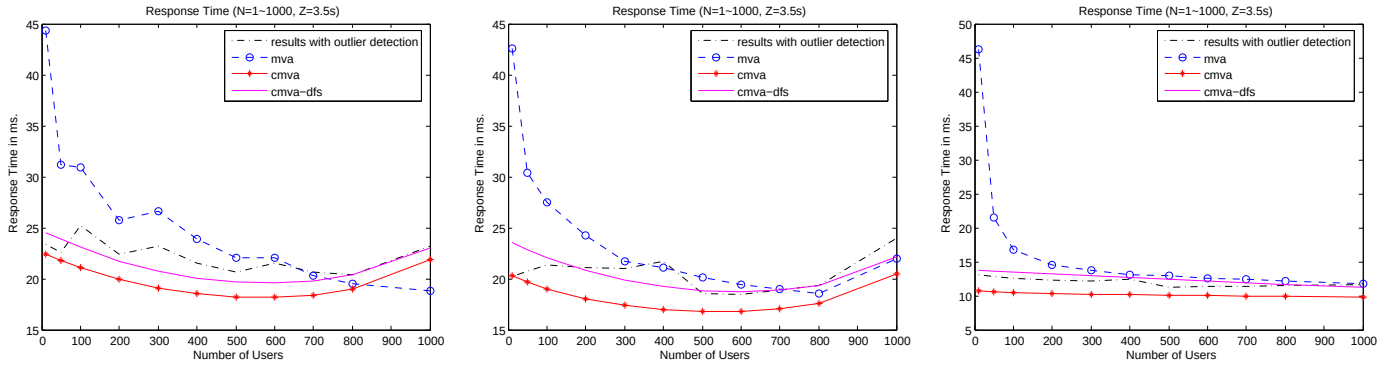
where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$-quantile of a unit normal variate (see Chapter 13 in [7]). In our tests, we choose $\alpha = 0.1$ (90% confidence interval), which is a typical value.

We now provide a couple of examples that demonstrate the degree to which these outliers can affect the average response time (see Table III). We calculate the average and the standard deviation for two classes (other classes also have similar results). As can be seen from the table, the standard deviations are relatively large compared with the corresponding averages. In other words, dispersion from the average is relatively large. The outlier region per class can be seen in the last columns in Table III. Similar results are also observed for other classes in TPC-W. In Table IV, we see that the 2.70% of response times that are outliers result in an increase of the average response time by 102.50% with $N = 1000$. In our experiments, we



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

Fig. 5: Service rate of FES with different mixes



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

Fig. 6: Predicted and measured R with different mixes and $Z = 3.5$ seconds (outlier detection enabled)

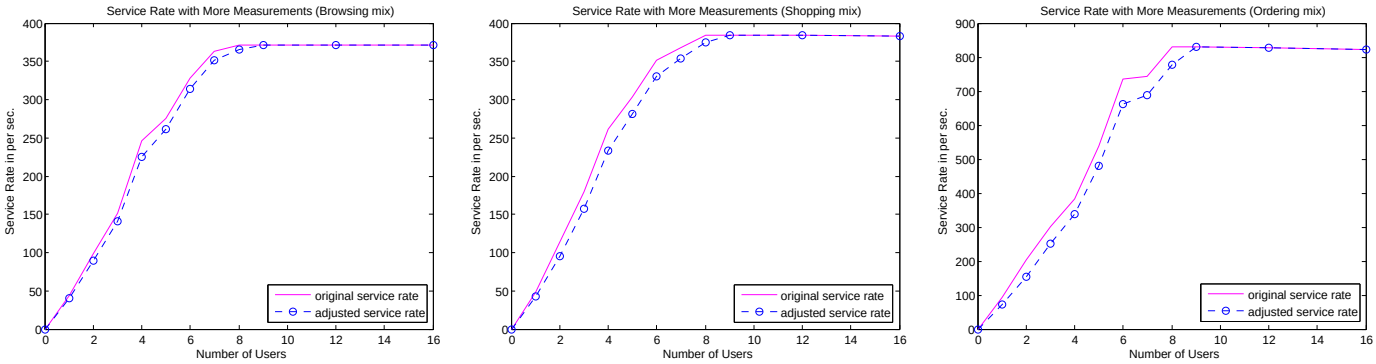
also found that the outlier phenomenon became worse as the workload increased.

V. Discussion

As can be seen in Figure 6, both the CMVA and the CMVA-DFS algorithms perform better than the multi-server MVA algorithm, especially under relatively light workload. In most cases, the multi-server MVA algorithm fails to predict the performance of the system, and it has 251.50% error in the worst case (in Figure 6c). The multi-server MVA algorithm is used for the queueing network model shown in Figure 3. We have suggested that this queueing model may be inapplicable due to violations of the product-form assumptions, and we verify this suggestion here. Therefore, the multi-server performance model is not validated for the systems considered in this paper.

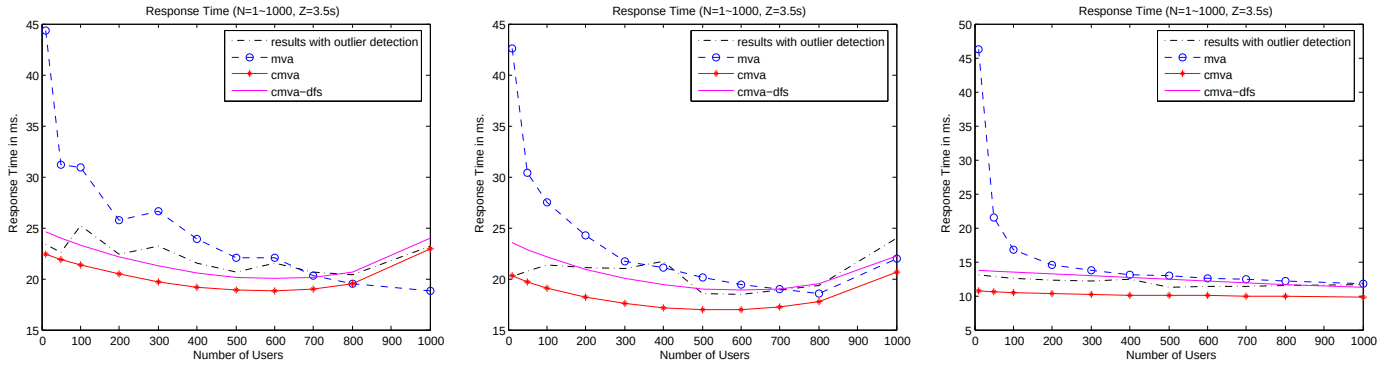
Compared with the CMVA algorithm, which constantly underestimates the average response time (by as much as 21.89% for the shopping mix), the CMVA-DFS algorithm produces better predictions. However, we observe that the CMVA-DFS algorithm overestimates the average response times under light load in Figures 6b and 6c. The underlying reason is the different proportions of requests in the different workload mixes. Compared to the browsing mix, the shopping and the ordering mixes both have a larger proportion of user interactions which require more operations at the I/O device, consequently, the disk demand is a greater component of the overall service demand. However, our approximation may overestimate the impact of DFS in the CPU under the assumptions in Section III-B. This can result in overestimating service demands under light load, and consequently the response times.

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. To illustrate the effects of generating the service rate curve from a limited number of measurements, we present three additional curves in Figure 7. These curves have more measured service rates compared with those in Figure 5 - the original service rates from one to eight are all obtained from measurements. In addition, we use these new curves to parameterize the CMVA and the CMVA-DFS algorithms - experimental results can be seen in Figure 8. Compared to the results in Figure 6, we do not observe any significant difference. The accuracy of our performance model is not affected by the relatively



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

Fig. 7: Service rate with more data from measurements



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

Fig. 8: Predicted and measured R with different mixes and $Z = 3.5$ seconds

coarse service rate curves. It is not clear that this is true in general, it may be that if the curve is generated more coarsely, the accuracy of APEM is affected. Thus, we suggest that the number of points should stay at a reasonable level to at least represent the shape of the curve, *i.e.*, choosing some key points on the curve. For instance, our testbed has eight cores, so that 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, and it may not work well in some cases when the assumptions discussed in Section III-B are violated.

VI. 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.* [24], [25] applied a regression-based approximation of the CPU demands of online web transactions. 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.* [10] also applied linear regression and a maximum likelihood technique to parameterize a performance model for an industrial ERP system. However, regression techniques suffer from the well-studied problem of multicollinearity (see Chapter 15 in [7]), which can lead to unreliable predictions for demands and very wide confidence intervals for predicted demands. Kalbasi *et al.* [9] and Mi *et al.* [18] have shown that the accuracy of regression-based techniques critically depends on the quality of monitoring data used in the regression analysis.

Zhang *et al.* [26] showed that it is not true that the service demand is load-independent for modern processors with DFS and HTT features. Their study demonstrated that the CPU demand can be modeled 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.* [11]. Using various workloads, they showed that their estimation method is more accurate. It is an interesting approach to explore the load-dependent behaviors 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.

Seidmann *et al.* [22] presented an approximation approach to avoid convergence problems of the load-dependent MVA algorithm with multi-server nodes. Chen *et al.* [5] applied this approach to a modified multi-class MVA algorithm to analyze a performance model for a multiprocessor system. In the approximation, a queueing node (with service demand D) which has K servers is replaced by two tandem queues. The first queue is a single-server node with service demand D/K, and the second one is a delay centre with delay $D(K-1)/K$. Experiments have shown that this approximation produces small errors under intermediary workloads [15]. However, it only works well for queueing models where the service demand of one server is constant.

VII. Conclusion

To address performance modeling for a multi-core computer system which cannot be modeled as a product-form queueing network, we proposed to solve a flow-equivalent aggregated model using a CMVA-DFS algorithm, where service demands are adjusted to compensate for the effects of DFS. In order to illustrate the applicability of our performance model, we applied it together with the CMVA-DFS algorithm in a TPC-W case study. The performance metrics (average response times) are validated with the experimental results. We have shown that the CMVA-DFS algorithm outperforms the CMVA algorithm and the multi-server MVA algorithm in the case study. We also discussed the errors generated by our performance model and its limitations.

There are several potential directions for future work. Firstly, we would like to adapt APEM for open or mixed (a mixture of open and closed) queueing network models. Secondly, to date we have only used benchmarks to generate artificial workloads for performance evaluation. It is an interesting research direction to validate APEM in real software 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.

References

[1] I. Ben-Gal. Outlier detection. In *Data Mining and Knowledge Discovery Handbook*, pages 131–146. Springer, 2005.

[2] G. Casale. A note on stable flow-equivalent aggregation in closed networks. *Queueing Systems*, 60(3-4):193–202, 2008.

[3] 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):1040–1053, 2012.

[4] G. Casale and G. Serazzi. Stabilization techniques for load-dependent queueing networks algorithms. *Communication Networks and Computer Systems*, pages 127–141, 2006.

[5] 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):299–311, 2008.

[6] T. Horvath. TPC-W J2EE implementation (University of Virginia), 2008. "http://www.cs.virginia.edu/∼th8k/downloads/", last accessed May 2013.

[7] R. K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John-Wiley, 1991.

[8] JBoss Community. JBoss application server website, 2004. "http://www.jboss.org/", last accessed May 2013.

[9] A. Kalbasi, D. Krishnamurthy, J. Rolia, and S. Dawson. DEC: service demand estimation with confidence. *IEEE Transactions on Software Engineering*, 38(3):561–578, 2012.

[10] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 48–48, 2009.

[11] 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*, page 47. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[12] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, pages 1–8. USENIX Association, 2010.

[13] W. Magro, P. Petersen, and S. Shah. Hyper-threading technology: Impact on compute-intensive workloads. *Intel Technology Journal*, 6(1), 2002.

[14] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1), 2002.

[15] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice-Hall, Upper Saddle River, NJ, USA, 2002.

[16] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy. *Performance by Design: Computer Capacity Planning By Example*. Prentice-Hall, Upper Saddle River, NJ, USA, 2004.

[17] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: symptoms, causes, and new models. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 265–286, 2008.

[18] 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 NOMS'08 IEEE*, pages 216–213, 2008.

[19] MySQL AB. MySQL community website, 2012. "http://www.mysql.com/downloads/mysql/", last accessed October 2012.

[20] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230, 2006.

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

[22] A. Seidmann, P. J. Schweitzer, and S. Shalev-Oren. Computerized closed queueing network models of flexible manufacturing systems. *Large Scale Systems*, 12(1):91–107, 1987.

[23] W. J. Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[24] 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):197–211, 2008.

[25] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of ICAC'07, Fourth International Conference*, pages 27–27, 2007.

[26] Z. Zhang, S. Li, and J. Zhou. Estimate load-dependent service demand for modern cpu. *Information Technology Journal*, 12:632–639, 2013.