A Capacity Planning Process for Performance Assurance of Component-Based Distributed Systems [*]

Nilabja Roy, Abhishek Dubey, Aniruddha Gokhale, Larry Dowdy
Dept. of EECS, Vanderbilt University, Nashville, TN 37235, USA
Contact: nilabjar@dre.vanderbilt.edu

ABSTRACT

For service providers of multi-tiered component-based applications, such as web portals, assuring high performance and availability to their customers without impacting revenue requires effective and careful capacity planning that aims at minimizing the number of resources, and utilizing them efficiently while simultaneously supporting a large customer base and meeting their service level agreements. This paper presents a novel, hybrid capacity planning process that results from a systematic blending of 1) analytical modeling, where traditional modeling techniques are enhanced to overcome their limitations in providing accurate performance estimates; 2) profile-based techniques, which determine performance profiles of individual software components for use in resource allocation and balancing resource usage; and 3) allocation heuristics that determine minimum number of resources to allocate software components.

Our results illustrate that using our technique, performance (*i.e.*, bounded response time) can be assured while reducing operating costs by using 25% less resources and increasing revenues by handling 20% more clients compared to traditional approaches.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory, Algorithms, Performance

Keywords

Multi-tier applications, performance estimation, service deployment

1. INTRODUCTION

A common requirement for large enterprise software systems such as (*e.g.*, eBay, Priceline, Amazon and Facebook), is high assurance of performance (*e.g.*, response time) and service availability to their users. Without such assurances, service providers of these applications stand to lose their user base, and hence their revenues.

High assurance of performance and availability to users in return for fees are typically specified as service level agreements (SLAs) between the user and the service provider. A straightforward approach to addressing the high assurance challenge is to over-provision resources which should be avoided due to increased costs. Addressing the high assurance problem without unduly affecting revenues reduces to addressing the capacity planning problem that can honor the SLAs across the user base (which often is on the rise in case of social networking portals).

Effective capacity planning requires an accurate understanding of application behavior. One approach is to develop analytical models of the application that can estimate the resource requirements and performance of the application. It is important, however, that the analytical model be reflective of real system behavior since it dictates the performance assurance of the multi-tiered application.

For example, if the model is optimistic, (*i.e.*, it estimates the average response time lower than the actual), then the capacity planning will result in lesser resources causing resource overloads and a violation of assurance of performance and availability. On the other hand if the model is pessimistic (*i.e.*, it estimates response times as higher than the actual), then the users will have assured performance but the system will use up more resources than actually needed, which is detrimental to the service provider.

Prior work based on analytical techniques and profiling to build models of multi-tiered web portals [15,19,21,23,24] exists but these efforts have not accounted for increased system activity, such as page-faults which occur with increased load, which is a frequent phenomenon. Moreover, the emerging trend towards multiple processors/cores has also not been considered by most of these works. Finally, resource allocation [6, 7], which is a key issue in capacity planning, has previously been investigated at the granularity of an entire tier-level. However most of modern multi-tiered systems are made up of finer-grained software components using com-

[*] This work has been supported in part by NSF SHF/CNS Award #0915976. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

ponent based technology such as .NET, J2EE etc. Thus resource allocation using these finer grained components is possible and might provide better results in minimizing the number of resources used.

This paper develops and presents a two-stage, design-time, hybrid capacity planning process that systematically combines the strengths of analytical modeling, profiling, and allocation heuristics in a novel framework called Modeling and Analysis using Queuing, Placement and Replication Optimizations (MAQ-PRO). The MAQ-PRO process hinges on a component-based structure of multi-tiered applications.

In the first stage, a profile-driven analytical model of the system is developed that can accurately estimate system performance even at high loads (which is a key factor that must be considered). The second stage uses this model as input to a replication and allocation algorithm that computes a deployment plan for the software components, which minimizes and efficiently utilizes resources.

To showcase our approach, we use a running example of a real-world, representative, multi-tiered component-based system called Rice University Bidding System (RUBiS) [1]. It is a prototype of an auction site that mimics eBay.

The rest of the paper is organized as follows: Section 2 presents the two-stage process provided by the MAQ-PRO framework; Section 3 presents an empirical validation of the replication and placement algorithm for the RUBis web portal case study; Section 4 compares our work with related work; and Section 5 presents concluding remarks.

2. MAQ-PRO PROCESS

This section details the MAQ-PRO process starting with the problem description. Formally, we state the capacity planning problem as follows: Suppose the multi-tiered application consists of a set of k services $\{S_1, S_2, ...S_k\}$. Each service is composed of software components, where a component C_{ij} is the j^{th} component in the i^{th} service. The target workload is given by either the arrival rate, λ, for each service $\{\lambda_1.....\lambda_k\}$, or the concurrent number, M, of customers $\{M_1, M_2, ...M_k\}$. The SLA gives an upper bound on the response times of each of the k services $\{RT_{sla,1}...RT_{sla,k}\}$. The objective is to find the minimum number of nodes to deploy the application on such that the SLA requirements of users are honored (thereby providing high assurance of performance and availability) while ensuring that resource usage is balanced.

We have developed a two stage framework called MAQ-PRO shown in Figure 1 to solve the capacity planning problem. Two stages were deemed necessary since deployment of components belonging to the services comprises of node allocation and balancing resource usage, which in turn depends on obtaining an estimate on the performance bounds of individual components. This dependency led us to separate the process of performance estimation from that of deployment planning.

We envision capacity planners using the techniques developed for Stage 1 to profile individual components and determining their resource requirements. Thereafter, different application scenarios can be analyzed, and using a base performance model, an application-specific analytical model can be developed that can accurately estimate the performance requirements of the system. In Stage 2, planners will use this analytical model as input to a component placement heuristic we developed that will result in a deployment plan which ensures minimum use of resources, which in turn provides assurances of performance and availability to the users.

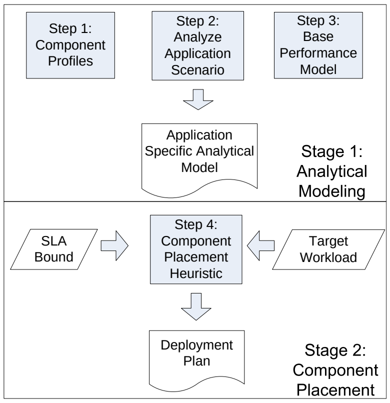

Step 1: Component Profiles
Step 2: Analyze Application Scenario
Step 3: Base Performance Model
Application Specific Analytical Model
Stage 1: Analytical Modeling
SLA Bound
Step 4: Component Placement Heuristic
Target Workload
Deployment Plan
Stage 2: Component Placement

Figure 1: The MAQ-PRO Process

We will now describe the two stage MAQ-PRO process using RUBiS as the guiding example. We consider three types of user sessions (visitor, buyer, and seller) provided by RUBiS, and a client-browser emulator for user behavior.

2.1 Stage 1: Estimating System Performance via Analytical Modeling

Component placement searches for the optimal/best combination of the components in the given nodes. Each particular combination will influence the application performance such as response time. Thus the combination with the best performance and the least resource used should be chosen. However for a large system, finding out the performance of every component combination is hard. There needs to be an analytical model which can accurately predict the application performance given any combination of the components in the machines.

Queuing theory enables the development of such performance models. The advantage of a queuing model is that armed with minimal profile data, it can estimate the performance characteristic of the application given any particular combination of the components in the various machines. These estimates can then be used to select the best component placement. However, the performance estimation need to be very accurate. Otherwise, when the application is deployed and if there are errors in the performance estimation then SLA bounds will be violated at production time.

2.1.1 Problem: Accuracy Of Performance Estimation Under Varying Hardware Configuration And Application Behavior

In,our previous work [10] we showed how a queuing model used in related research [19, 21, 23, 24] does not provide accurate response time estimates when the client population increases.

To validate our claim, a multi-class closed queuing model is developed for RUBiS as shown in Figure 2 for a scenario comprising two machines. One machine acts as the joint web server and business tier server while the other operates as the database server.

A queue is modeled for each of the resources in the machines, *i.e.*, CPU and disk. Each service r provided by the application is mapped as a separate job class. The terms service and class will be used interchangeably in the rest of

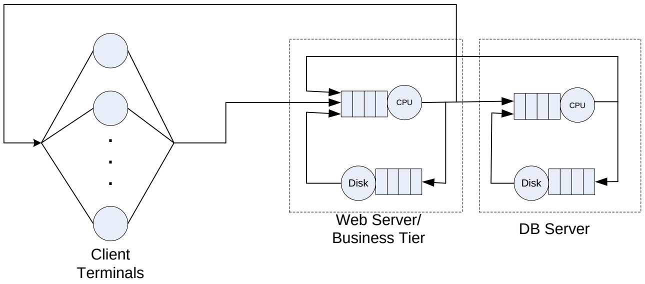


Figure 2: Queuing Model of RUBiS Scenario

Configuration	Configuration Details
Configuration 1	Four 2.8 GHz Xeon CPUs, 1GB ram, 40GB HDD
Configuration 2	Quad Core 2.4 GHz Xeon CPUs, 4GB ram

Table 1: Machine Configurations Used

the paper. The main parameter to this model is the service demand of each component which is the resource time taken to complete a single job. The basic function of a component comprising each service can be profiled to measure the service demand of the component. Once the component is placed in a given machine, the service demand of the component will be imposed on the devices of the machine.

The above queuing model is used to predict the response times of various services of RUBiS under two different hardware configurations. The hardware configurations are given in the Table 1:

Two sets of experiments are run in each machine configuration, one when a single service is run another in which multiple services(class) are run. The former exhibits the performance of each component running in isolation while the latter gives an idea of the effect of collocation of components. In both cases, we use the above queuing model to predict the response time. The parameters (service demand) of the queuing model depend upon the machines on which the application runs. The service demands are thus computed by profiling each single component on the different machines by running with a single client in the system. In previous work[14], it was seen that CPU activity of RUBiS increases with load compared to memory and bandwidth which remain fairly constant. Thus CPU is mainly considered here.

Figure 3a shows the response time when a single service of RUBiS is run with Configuration 1. Similar behavior is also seen when multiple services are run, as shown in Figure 3. Here we reproduce two of the services “SearchByCategory” and “SearchByRegion” which have higher response times. In this experiment around 12 services are running each service having 3 components. The other services also incur similar estimation errors.

Figures 4a, 4b, 4c show the response time of the same above services when multiple services are running together in Configuration 2. The model prediction is also shown. Strange behavior is seen in this experiment, as shown in Figure 4c where the CPU is only loaded till around 30%. After this point, as load increases the response time shoots up even though the CPU is underloaded. The memory and bandwidth also remains much below its capacity(not shown here for lack of space). It is evident that there is some other bottleneck in the system which causes the response time to shoot up. It could be due to software contention. The queuing model understandably cannot predict this behavior since the ”invisible bottleneck” is not modeled. Thus it estimates the CPU utilization to increase with load while in reality it saturates around 30% as shown in Figure 4c.

Finding the root cause of this bottleneck is hard since it might require investigating immense amount of code and analyzing various scenarios. In a real world scenario it could also be caused by third-party libraries, the code for which may not be available. Stewart et. al. [16] discuss such a scenario where anomalies in an environment affect performance. In such a scenario, a basic queuing model becomes increasingly erroneous and cannot be relied upon for proper prediction.

A remedy would be to profile the system with different workload and create statistical regression models. But such an approach will not help us in predicting the performance when components are arbitrarily placed in different combinations in the machines since this will require us to profile the application using every combination of the components which is clearly not possible. The next section details the solution approach followed in this work.

2.2 Solution: Profile driven Regression based Extended Closed Queuing Network

This section discusses the details of the modeling techniques developed by enriching basic closed queuing models with statistical regression models to come up with increased accuracy in estimating application performance.

In our approach we come up with the following steps to produce better models: (a) Profile individual components, (b) Create regression models, and (c) Extend queuing models with regression models

This helps us in estimating the performance of multiple components running together in the same machine using profile data of individual components. On one hand it leverages the strength of regression models where unique scenarios or environmental effects are captured through profiling and on the other hand uses the strength of queuing models which enables the performance estimation of multiple classes (or collocated components). The above approach ensures that the profiling is kept to no more than what is absolutely required and also leverage existing queuing models to estimate multi-component behavior which can be used for component placement decisions.

2.2.1 Modeling Increased System Activity

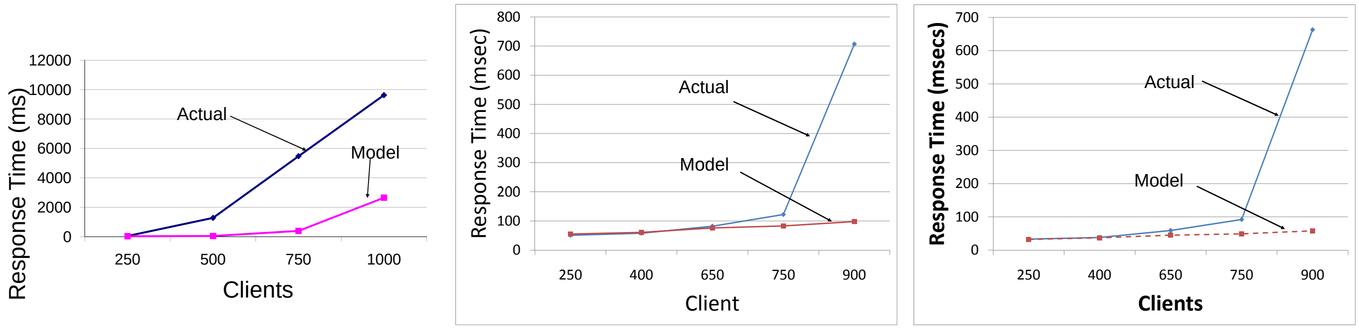
The Figures 3a, 3b and 3c show that there is increased error in model prediction at high load. This section discusses the probable reasons for such error and comes up with solutions to address them.

Queuing models can be efficiently solved using approximate Mean-Value Analysis (MVA) algorithm [9]. The main equation that is used to compute response time is given by

$$R = D_{ir} + n_i \times D_{ir} \tag{1}$$

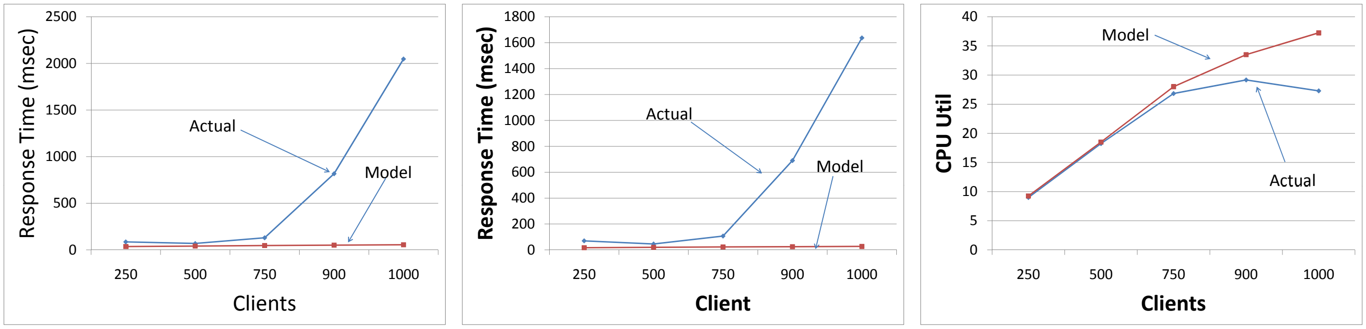
where D_{ir} is the service demand of service r on device i and n_i is the number of waiting jobs on device i. The service demand gives the actual resource time used by a component while processing the job. At high load, additional activity in the machine due to system work including context switches, paging, swapping etc also adds to the response time. This excess system activity is not accounted by Equation 1.

To further investigate this intuition we measure the number of context switches that occur per second as client popu-



(a) Single Service (b) Multi Service: SearchByRegion (c) Multi Service: SearchByCategory

Figure 3: Comparison of Analytical vs Empirical Data



(a) Multi Service: SearchByRegion (b) Multi Service: SearchByCategory (c) CPU Utilization

Figure 4: Comparison of Analytical vs Empirical Data In Configuration 2

lation is increased (measured using Configuration 1 as given in Table 1). This data is plotted in Figure 5. It can be seen that the number of context switches per second steadily increases with increase in client population. Since context switch is one type of system activity, it clearly shows that the amount of system activity increases with clients. The challenge now is to capture and quantify this increased system activity in terms of resource usage so that it can be added to the MVA analysis which will then produce better estimation of performance parameters.

Next we measure the total CPU utilization per job as we increase client population. This measurement is done by capturing the total CPU utilization for the lifetime of the experiment and dividing it by the total number of jobs completed in the same interval. The observed total CPU utilization per job is shown in Figure 5 along with the context switches per sec. It is seen that the CPU utilization per job steadily increases along with the number of context switches per sec and becomes steady after some time. Initially, at very low load (single client) there is nearly zero context switch/sec. The CPU utilization/job is also very less and matches with the service demand value. Consequently it can be deduced that as system activity increases the excess CPU utilization per job is due to additional system activity. Obviously such effect must be accounted for in a performance model. However, traditional queuing models do not account for this behavior.

To overcome this limitation we define a term ”Overall Service Demand” (OSD) which is defined as the total resource time required to complete a single transaction. Thus the CPU utilization shown in Figure 5 is actually the OSD for the concerned service. As shown in Figure 5 the OSD has the potential to vary with load since it is the sum of the service demand and resource usage due to system activity.

Overall service demand (OSD) can be measured using the service demand law [9]. The service demand law is given as $D_i = U_i/X$ where D_i is the service demand on the i^{th} device, U_i is the utilization of the i^{th} device, and X is the total number of transactions/sec or throughput of the system. When the service demand law is used at high load it returns the OSD which is a sum of the service demand and the resource time spent due to system activity. The OSD can thus be obtained for different client population by measuring the device utilization and the throughput of the services while client size is varied. The measured values are then used with the above law to obtain the OSD.

We empirically profiled each service hosted by the RUBiS web portal by varying the client size from an initial small value to a large value. Here we assume that individual components (services) of a large, multi-tiered system are available for unit testing and profiling. We measured the processor usage and the number of successful calls for each client population size. The service demand law is then used to compute the overall service demand for each client size.

As seen in Figure 5, the overall service demand remains steady at low utilization (≤ 10) and then follows a near linear increase till around 80% utilization or 350 clients. The linear rise can be attributed to the increase in system activity as clients increase. Since each client represents a thread in RUBiS, consequently, an increase in the number of clients increases the number of threads.

This behavior is better understood from the number of

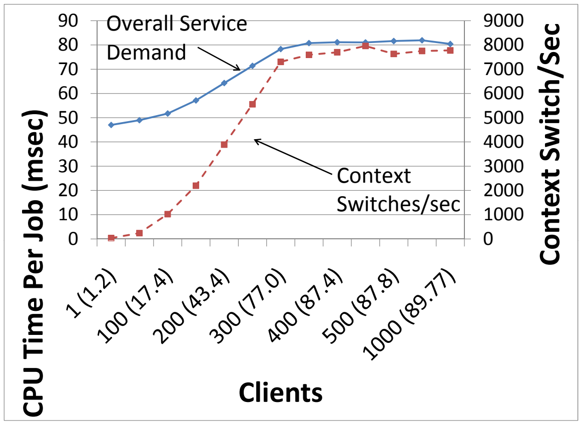


Figure 5: Overall Service Demand

context switches as utilization and clients increases. There is negligible context switching for low number of clients but increases linearly with clients until 350 clients when it becomes steady. At 350 clients, the service demand also stabilizes because the device (*e.g.*, CPU) utilizations are close to saturation (greater than 90%) and there is not much scope for any increase in system activity. We have observed similar behavior in the other services of RUBiS.

Based on these insights, the overall service demand is modeled as a load-dependent function of processor utilization which is piecewise linear. To empirically obtain accurate demand functions, the Polyfit tool provided in the Matlab Curve Fitting Toolkit is used. The resulting function which represents the overall service demand for the `SearchByRegion` service is given by:

$$OSD_{sr}(U) = \begin{cases} 48 & \text{for } U < 8 \\ 0.4264 \times U + 45.1062 & \text{for } 8 <= U <= 85 \\ 81.62 & \text{for } U > 85 \end{cases} \quad (2)$$

and the function representing the service demand for the `SearchByCategory` service is given by:

$$OSD_{sc}(U) = \begin{cases} 28 & \text{for } U \leq 5 \\ 0.0457 \times U + 24.94 & \text{for } 5 <= U <= 84 \\ 52.06 & \text{for } U \geq 84 \end{cases} \quad (3)$$

The coefficient of determination, R^2, value for the linear fit is 0.99 for both equations indicating very good fits. Capacity planners using MAQ-PRO should adopt a similar approach to obtain accurate functions for overall service demands of individual services belonging to their applications.

The MVA algorithm used is now modified to include usage of the overall service demand instead of the original service demand which represents the actual resource time used by a transaction. Thus Equation 1 is replaced by the following:

$$R = OSD_{ir}(U) + n_i \times OSD_{ir}(U) \quad (4)$$

where OSD_{ir} is the overall service demand for the r^{th} class on the i^{th} device. So the single constant value of service demand is replaced by overall service demand which takes into account the system activity in the machine. As the overall service demand increases with utilization, the number on the device(n_i) also increases and both controbute to the response time to increase.

Using the above version of MVA, we validate the response time prediction of the model against actual measured response time under a single processor machine and is shown in Figure 6. It can be seen that for single processor machines, our extended MVA can nicely approximate the response time.

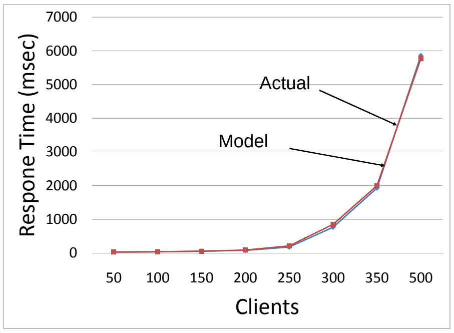


Figure 6: Response Time in Single Processor Machine

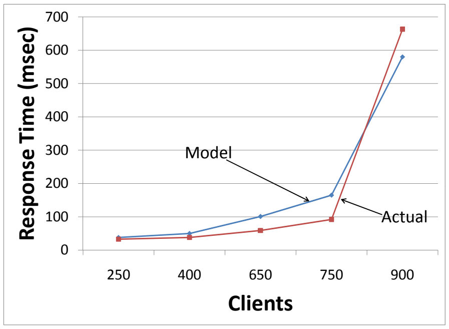


Figure 7: Response Time in Multiple Processor Machine

2.2.2 Modeling Multiprocessor Effects

Due to the increasing availability and use of multi-processors and multi-cores for large scale applications, such as web portals, existing closed queuing network models must model multi-processor effects on performance. We use the extended version of MVA explained in Section 2.2.1 to validate the prediction under hardware Configuration 1 as given in Table 1. Figure 7 compares the model estimation with empirical measurement and shows that there is still some gap in the estimation which is investigated in this next section.

Typically multiple-server queuing models are solved by considering each multiple server as a load dependent server [9] and computing the probability mass function of the queue sizes for each server. The mass function can then be used within MVA to calculate the total expected waiting time that a customer experiences on a server. This approach, however, significantly increases the complexity of the MVA solution. There have been attempts in recent research [17] in which a simple approximate method is presented that extends MVA to analyze multiple-servers. In [17], the authors introduce the notion of a *correction factor,* which estimates the waiting time. When a transaction is executed on multiprocessor machines, the waiting time for each transaction on the processor is taken to be the product of a constant factor, the service demand, and the average number of waiting clients as captured by the following formula:

$$R(N) = SD + c \times SD \times n \tag{5}$$

where $R(N)$ is the response time of a transaction when there are a total of N customers in the system, SD is the service demand of the job, n is the average number of customers waiting on the device, and c is the correction factor to compute the waiting time. In their work they theoretically compute the value of the correction factor. [17] also considers a constant service demand and thus Equation 5 need to be adjusted by using Overall Service Demand instead of service demand to incorporate increased system activity at high load. Equation 6 shows the revised version including overall service demand (OSD).

$$R(N) = OSD(U) + c \times OSD(U) \times n \tag{6}$$

We surmise that such a correction factor will depend on a number of factors, such as the domain of the operation, and the service time characteristics for the underlying hardware, the cache hit ratio, memory usage levels, memory sharing etc. Therefore, the correction factor will vary with each different scenario and need to be profiled on the particular hardware. We now describe how we found the correction factor for the RUBiS example.

Capacity planners using the MAQ-PRO process should adopt a similar approach for their applications. The data needed to compute the correction factor can be extracted from the same experiments done to estimate the OSD as mentioned in Section 2.2.1. Thus there is no need to conduct additional experiments and a single experiment will suffice for both the OSD and the correction factor. Our approach again is to profile individual components and then estimate the expected performance when any combination of the components are placed in the machines.

Referring to Equation 6, the value of the overall service demand $OSD(U)$ can be found using the profile-based curve fitting approach explained in Section 2.2.1. The average number of customers waiting on the CPU, n, is obtained by using standard system monitoring tools. The response time for each transaction, $R(N)$, can be obtained from the application logs or by time-stamping client calls. The only unknown in Equation 6 is the correction factor, c, which can be obtained by solving the equation.

We ran a number of experiments for different classes of services supported by RUBiS with different client population sizes and the variable n was monitored. $R(N)$ was obtained from the RUBiS logs. The load-dependent service demands, $OSD(U)$, were obtained from Equations 2 and 3. The correction factor was then computed using Equation 6, which is presented in Table 2 for two different services in RUBiS for a 4 processor machine. Figure 8 shows the comparison of the empirically obtained correction factor to the one proposed by Suri. It clearly shows that the actual correction factor is much different and depends upon the specific scenario.

Table 2 presents the experimental values and the computation for the correction factor with different client population for the two main services in RUBiS. The inverse of the correction factor is given in the rightmost column of the table. It is termed as CI. It can be seen that the correction factor varies with clients or processor utilization.

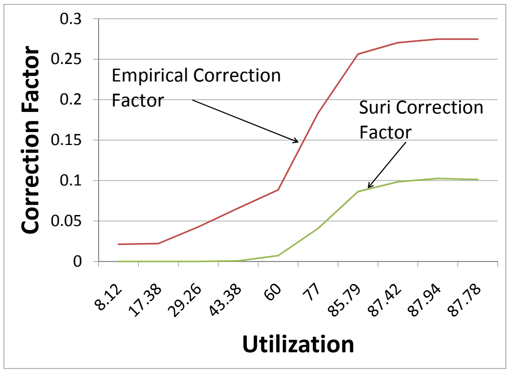


Figure 8: Comparison of Empirical Correction Factor with Suri Proposed

Service	Clients Name	Service Demand (msec)	Avg Waiting	Response Time	Corr. Factor	CI CI
Search	100	51.71	2.00	54	0.022	45.16
ItemsByReg	150	57.12	2	62	0.043	23.40
	200	64.29	3	77	0.066	15.17
	250	71.4	5	103	0.089	11.29
	300	78.3	10	222	0.183	5.45
	350	80.78	40	909	0.256	3.90
	400	81.12	86	1968	0.27	3.69
	500	81.62	185	4232	0.275	3.64
Search	100	51	2	54	0.029	34.00
ItemsByCat	150	31.25	2	34	0.044	22.73
	200	33.45	2	37	0.053	18.85
	250	35.6	2	40	0.062	16.18
	300	38.38	3	47	0.075	13.36
	350	41.28	4	58	0.101	9.88
	400	43.16	5	73	0.138	7.23
	450	46.14	8	116	0.189	5.28
	500	50.88	34	513	0.267	3.74

Table 2: Correction Factors for Various Services

Since the correction factor actually represents the multiprocessor effects on performance, it should be dependant on the number of processors in the machine. To validate our hypothesis, we configured the machine to use different number of processors and repeated the experiment with 1 and 2 processors, respectively. Figure 9 shows the value of CI with clients for the service "SearchByCategory". Similar results were obtained for other services but are not shown due to space constraints.

The value of CI is interesting. It has a very high value with less load but slowly converges to a steady value at high load. The steady value appears to converge to the number of processors in the system. It can also be seen that the variation in the factor increases with increase in processors. Higher values of CI (*i.e.*, lower value of the correction factor) improves the response time as seen from Equation 6. This observation indicates that the correction factor could also be indicative of the inherent optimizations such as caching that occur in the system.

This hypothesis needs further investigations and will become part of our future work. It also tells us that at high load there may not be much scope of optimization and the system behaves like a straightforward fluid flow system and can be modeled using variations of fluid flow modeling techniques as done by many recent work. [3, 8, 12]

The value of CI for each client population is averaged over all the services. It is then approximated against processor utilization. A piecewise linear function is developed to express CI as a function of utilization which is calculated using polyfit function in Matlab and is given by

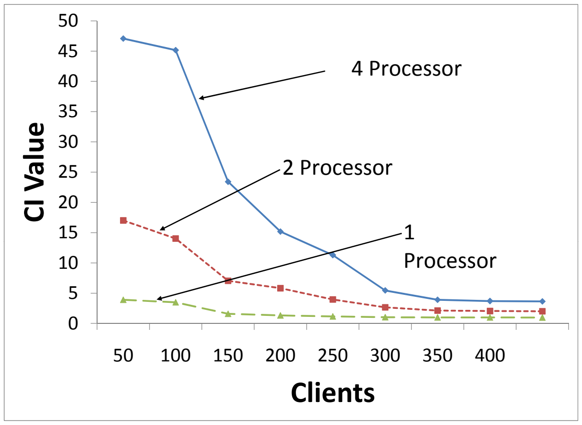


Figure 9: Inverse of Correction Factor (CI)

Algorithm 1: Modified Mean Value Analysis

```
Input:
    $R$ Number of Job Classes
    $K$ Number of Devices
    $D_{i,r}$ Service Demand for $r^{th}$ job class on $i_{th}$ device
    $N_r$ Number of clients for $r^{th}$ class
    $U_r$ Utilization of the $r^{th}$ device
    $X_r$ Throughput of the $r^{th}$ device
Output:
    Response Time $R \leftarrow$ vector containing response time for all
    classes of jobs
begin
    // Run initial MVA with lowest service demand
    while $Error > \epsilon$ do
        // Initialization ....
        for $r \leftarrow 1$ to $R$ do
            for $i \leftarrow 1$ to $K$ do
                $\mathbf{D_{i,r}} = \mathbf{OSD_{i,r}}(\mathbf{U_r})$ // Call function for Service
                    Demand with device utilization as parameter
                $R_{i,r} = D_{i,r} \times (1 + \mathbf{CI(U_r)} \times n_r)$
            $X_r = \frac{N_r}{Z_r + \sum_{i=1}^{K} R_{i,r}}$
        // Error = Maximum Difference in Utilization between
           successive iterations
```

$$CI(U) = \begin{cases} -0.5632 \times U + 38.75 & \text{for } U <= 58 \\ -0.1434 \times U + 15.71 & \text{for } 58 < U < 85 \\ 3.69 & \text{for } U \geq 85 \end{cases} \tag{7}$$

Equation 7 is then used from within MVA algorithm to compute the response time in each iteration.

2.2.3 Modifying Mean Value Analysis Algorithm

As described in Section 2.2.1, we develop a multi-class closed queuing model for RUBiS as shown in Figure 2. An approximate MVA algorithm based on the Schweitzer [9] assumption can be used to solve this model and calculate performance values, such as response time, number of jobs in the system, and device utilizations for closed systems [9]. We developed an approximation to the original MVA algorithm as shown in Algorithm 1. Some details in the initialization phase are not shown due to space constraints.

The algorithm starts by assuming that the clients are evenly balanced across all the devices and then adjusts the clients in the various devices iteratively. In each iteration, the algorithm computes the number of clients on each device, response time, utilization and throughput of each job type. It continues this iteration until the error in the number of clients in each device reduces below a given minimum.

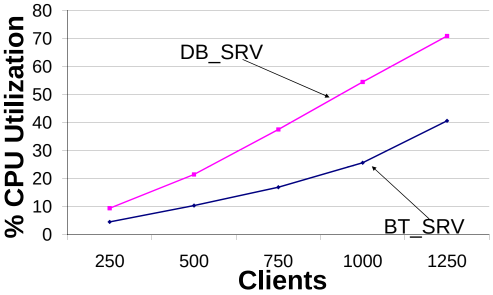


Figure 10: CPU Utilization

The boldface parts shown are the places where the original MVA algorithm is modified to include the functions for overall service demand and refined correction factor. The function $OSD_{i,r}$ represents the service demand function for r^{th} job class in the i^{th} device while function $CI(U_r)$ is the Equation 7. Both of these functions need device utilizations which is computed on every loop. They also need an utilization value which needs to be provided for the first iteration. For this reason, initially the first iteration is run using the lowest value of overall service demand for each service as given by Equations 2, 3 and the value of CI equal to the number of processors in the system.

2.3 Stage 2: Minimizing Resource Usage in Component Placement

Having developed accurate analytical models for the multi-tiered applications in Stage 1, the next task is to determine the number of resources needed to host the components of the different services with a goal towards minimizing the number of resources.

To address the next problem, it is important to understand the client behavior. For example, different kinds of client actions and the services they use will determine the overall workload on the multi-tiered application. Some services may impose more load compared to the others depending on which ones are heavily used by the user population. Accordingly it may become necessary to deploy multiple instances of the software components that implement the highly loaded services so that the total load can be balanced between different instances. An important question stems from determining which components need to be replicated for load balancing. This question must be accounted for while minimizing the total number of resources needed for all the components.

To highlight the need for load balancing, we reproduce Figure 10 from our earlier work [14] in which processor utilizations of two servers in RUBiS are shown. In the machine `DB_SRV`, only one component called `SearchItemsByCat` takes up 70% of processor time when the number of clients reaches around 1,300. At the same time, the other machine shown by Line `BT_SRV` is loaded only upto 40%. Thus, there is an imbalance in resource usage.

Resource allocation algorithms developed in prior research [6, 7, 14] cannot improve this situation since the single instance of component `SearchItemsByCat` takes up significant CPU. To overcome this limitation, a promising solution is to add a new instance of `SearchItemsByCat` component and distribute the load between the two machines. Consequently,

Item Size	% Slack			
	0 - 5	5 - 10	10 - 15	15 - 20
0 - 30	34.84	97.96	99.97	100
0 - 50	10.57	65.49	96.14	99.67
0 - 70	26.44	65.68	93.02	99.14
0 - 100	100	94.93	99.34	99.64

Table 3: Success Rate of Heuristics on Solvable Problems:Courtesy [13]

one of the components could then be placed onto `BT_SRV` so that the overall earlier load of $70+40 = 110$ can be balanced across the two servers (55 each). Such allocation makes it possible to handle more clients since now the utilization of both servers can be increased to around 70.

This evidence suggests that by replicating individual components and controlling the distribution of load on a component, we can balance the resource requirement of components in various nodes and thus minimize use of resources. In the remainder of this section, we will refer to the percentage resource required by a component as the size of the component. The challenge now is to determine the size of each component that will help in balancing the load and minimizing resources, which is a non-trivial problem [20].

The problem becomes more acute when trying to determine component placement at design-time, which require models that can accurately estimate the component size as well as performance of the overall application for a particular placement. We leverage Stage 1 of the MAQ-PRO process to obtain accurate estimates for each component.

We present our technique for determining the replication requirements and placement decisions for software components in the context of the different services offered by RUBiS. Capacity planners using MAQ-PRO should adopt similar stategy for their applications.

The lower bound on the total number of machines required for a web portal like RUBiS can be calculated from the expected processing power required in the following way:

$$\#ofmachines = \lceil Ld/m \rceil <= OPT, \tag{8}$$

where Ld is the total processing power required (sum of the CPU requirement of all the components) and m is the capacity of a single machine. OPT is the optimal number of bins required to fit the given items.

The problem of allocating the different components onto the nodes is similar to a bin-packing problem [5]. The machines are assumed to be bins while the components are items, where the items need to be placed onto the bins. It is well-known that the bin-packing problem is NP hard [20]. Thus, popular heuristics like first-fit, best-fit or worst-fit [5] packing strategies must be employed to determine the allocation. It has been shown that these heuristics provide solutions which require $(1.22 * OPT + 1)$ bins [5].

Our previous work [13] did an extensive study on the effectiveness of the different bin-packing heuristics under various conditions. We found that the size of items used in packing made a significant difference to the results generated by the heuristics as shown in Table 3. Here all quantities are mentioned in terms of percentages, *i.e.*, percentage of a single bin size. So an item size of 20% means that the resource requirement of a component is 20% of the total CPU time. The table shows the probability of finding an allocation of the given items onto the bins with different values of slack (difference between total bin capacity and total of all packed item sizes) and for different item size ranges.

For example, the entry of the third column and first row is 97.96%. This means that if there are items sized between 0 and 30% (row value) of bin size and slack between 5 to 10% (column) of bin size, then the chance of finding an allocation is 97.96%. This also means that if the item sizes are kept between 0% and 30%, then the heuristics can find an allocation using up to around 10% more space than the total item sizes. Thus, the expected number of machines required would be $\lceil 1.1 \times Ld/m \rceil$ which is less than $(1.22 * OPT + 1)$ as per Equation 8.

The above insights are used in the component replication and allocation algorithm developed for this paper. Our algorithm requires that component sizes be kept within 30% which means the component resource requirement is kept within 30% of total processor time. We satisfy this requirement by figuring out the number of clients that drive the utilization of the processor to 30% due to that component and allowing only these many clients to make calls on a single component instance. Such an approach can easily be implemented by a sentry at the server level that monitors the incoming user requests. Algorithm 2 describes the component replication and placement algorithm. It performs a number of functions as follows:

- **Capacity Planning:** It computes the number of nodes required for a target number of customers while minimizing the number required.
- **Load Balancing via Replication**: It computes the number of replicas of each component needed to distributed loads on the components and achieve balanced resource usage.
- **Component Placement**: It computes the mapping of the different components onto the nodes.

Algorithm 2 uses two subroutines, `Placement` and `MVA`. Placement places the components onto the machines by using the worst-case bin packing heuristic since it is known to balance load. MVA is the Mean Value Analysis algorithm that uses the enhanced analytical models developed in Stage 1 to accurately estimate performance characteristics of a closed queuing network. It returns the response time of the different transaction classes along with the utilization of each component and each machine.

Initially, the algorithm starts with a default set of components needed for each service, uses a tiered deployment, and assumes a low number of clients, say, 100 (Line 7). A 3-tiered deployment typically uses one machine per tier but Algorithm 2 starts with 2 machines to attempt to fit the application in lesser machines. The components of each type are placed in the respective machines. The algorithm starts by estimating the performance characteristics of the application and placing the different components onto the given machines (Lines 8 & 9).

Next, the algorithm enters an iterative loop (Line 11) increasing the number of clients with each iteration until the target number of clients is reached. At every iteration MVA is used to estimate the performance requirement (Line 12). If any component reaches 30% utilization (Line 13), then another instance of the component is created and initially placed in the same machine as the original. Then MVA is invoked to estimate performance and the components are again placed onto the nodes. Similarly, if at any point the response time of any transaction reaches the SLA bound (Line 19), then another machine is added to the required

Algorithm 2: Replication & Allocation

```
begin
    // Initially, use 2 machines in a tiered deployment

    // All business logic components in first machine

    // Database in second machine, Default Deployment Plan DP

    P = 2 // Initially 2 machines
    N = init_clients
    (RT,SU,U) = MVA (DP, N) // Compute Initial
        Component Utilizations
    (DP) = Placement (SU, P) // Find a placement of the
        components

    while $N < Target$ do
        (RT,SU,U) = MVA (DP, N)
        if $\exists i : SU_i > 30$ then
            Replicate (i); // Create New instance of
                Component i

            // Place new component on same machine as i
            (RT, SU, U) = MVA (DP, N) // Calculate new
                response time
            (DP) = Placement (SU, P) // Update
                Deployment Plan
        if $\exists i : RT_i > RT_{SLA}$ then
            // add new machine
            P = P + 1
            (DP) = Placement (SU, P) // find new
                placement
        N += incr // Increase Clients for next iteration
```

node set and the placement heuristic is invoked.

This iterative process continues until the target number of clients is reached. Since the heuristic is one of the popular bin packing heuristics and the components are kept within a maximum of 30% resource utilization, it is ensured that near-minimum number of resources will be used.

3. EVALUATION

This section presents results that evaluate the two stage MAQ-PRO framework. The results are presented in the context of the RUBiS example along two dimensions: the accuracy of the analytical models to estimate performance, and the effectiveness of the resource allocation algorithm to minimize the resources required while supporting increased number of clients, as well as balancing the utilization – which collectively are an indirect measure of high assurance in terms of performance and service availability to users.

3.1 Stage I Model Validation

Our analytical model have been enhanced by using equations 2, 3 and 6. The training data used to develop these equations is found by profiling each of the components separately. While validating our model, we use performance data when multiple components run together in the system. This will help us in understanding the prediction power of our models given an arbitrary set of components running in the same machine.

RUBiS has 10 service types for a typical browsing scenario consisting of item searches, user searches, viewing user comments, viewing bid history etc. Our objective is to check how well our model predicts the response time of each of the service types and the processor utilization of the machine when all such services are running. Each service is implemented using multiple components thus our training data greatly differs from the validation data. Our training data consists of performance measures when each single component runs while our validation data consists of performance measures when multiple components run together.

		Node Utilization			
Deployment	Response Time (msec)	Node 1	Node 2	Node 3	Node 4
Tiered	270	51.06	79.08	17.47	78.86
MAQ-PRO	353.5	87.32	57.41	65.04	

Table 4: Response Time and Utilization

Figure 11a shows the response time estimated by our model for the service, ”SearchByRegion” using machine Configuration 1 as given in Table 1. The estimation of the other services are also similar. It can be seen that our enhanced model is in close agreement with the empirical measurements till the number of clients equal to 900. Beyond that number, the error in our model increases slightly but still is close to the actual result. The error in estimation at such high load will not effect the overall capacity planning process since the system is saturated at this stage with CPU utilization nearing 90% and the operating region of the application need to be below this value to ensure SLA compliance. Figure 11b compares the CPU utilization predicted by the model versus the empirically measured CPU utilization. It can be seen that the model is in agreement with the empirical data for all client population size.

Figures 12a, 12b, 12c shows model estimation of the performance data in Configuration 2 using the extended models. It can be seen that the performance predictions are quite close and there is very little error.

3.2 Effectiveness of the Stage II Placement Algorithm

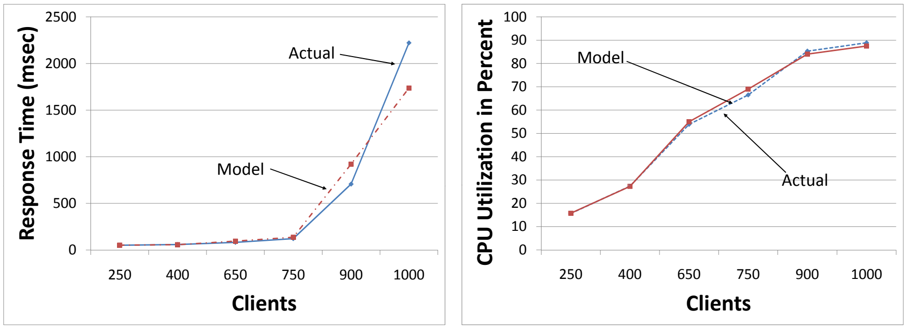
We now present results measuring the effectiveness of the MAQ-PRO Stage 2 placement algorithm. The evaluation tests the merits as follows:

1. Minimizing and Efficiently Utilizing Resources: In a traditional tiered deployment, each tier is considered atomic and hence all its functionality must be deployed together. In contrast, for a component-based system where services are implemented by assembling and deploying software components, it is possible to replicate and distribute individual components over the available resources. We argue that this flexibility can make better usage of resources compared to a traditional tiered architecture.

Figure 13 presents a number of scenarios in which the algorithm was evaluated. It compares the number of machines required to support a given number of clients for a range of client populations. Each client has a think time of mean 7 seconds with exponential distribution. The service times of the requests are also distributed exponentially. Even if the service times are non-exponential in the real world, the above models will produce good results due to the robustness of closed non-product-form queuing networks. By robustness, we imply that a major change in system parameters will bring about tolerable changes in computed parameters [2].

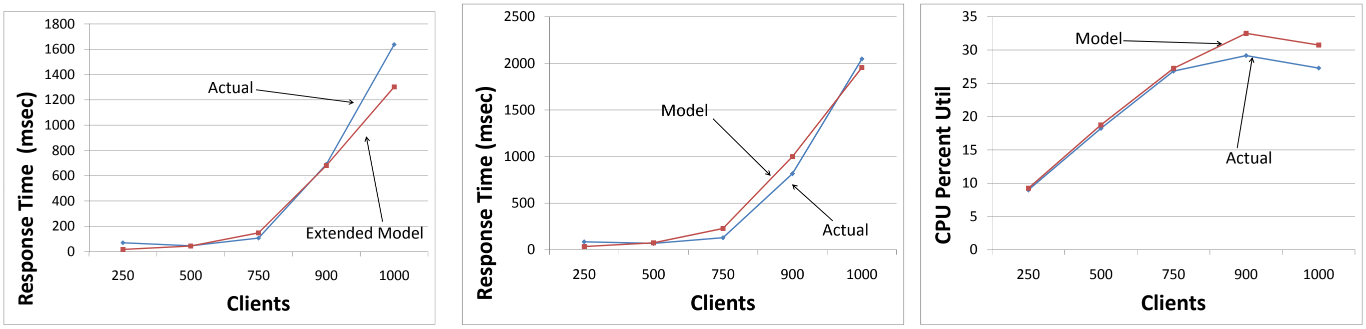
For every value of client population considered, the response time of the client requests remained within the SLA-prescribed bound of 1 second. It can be seen that for a majority of the cases our algorithm finds an allocation of the components that uses a reduced number of machines compared to the traditional tiered deployment.

Table 4 shows the response times and the utilizations of



(a) SearchByRegion Response Time (b) CPU Utilization

Figure 11: Model Validation for Configuration 1



(a) SearchByCategory Response Time (b) SearchByRegion Response Time (c) CPU Utilization

Figure 12: Model Validation for Configuration 2

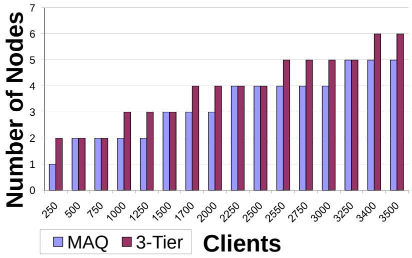


Figure 13: Node Usage in Tiered and MAQ-PRO Deployments

the different processors for one such scenario with a total client population of 2, 000. A tiered deployment requires 4 machines to serve 2, 000 clients, while MAQ-PRO requires only 3 machines – an improvement of 25%. The table clearly shows that in the tiered deployment, Node 3 is mostly idle $(17.47\% utilized)$. MAQ-PRO identifies idle resources and intelligently places components resulting in a minimum of idle resources.

Figure 14 shows the resulting allocation of the different components in the deployment of RUBis web portal using MAQ-PRO Stage II. Using multiple instances of components and distributing them in an intelligent way helps in effective utilization of available resources.

Figure 15 presents the coefficient of variance (CV) of the CPU usages for the three machines used in this experiment.

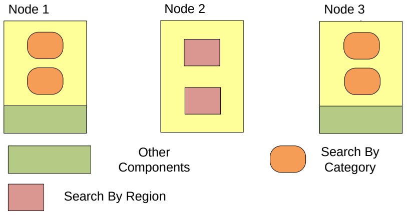


Figure 14: Component Placement : 2,000 Clients

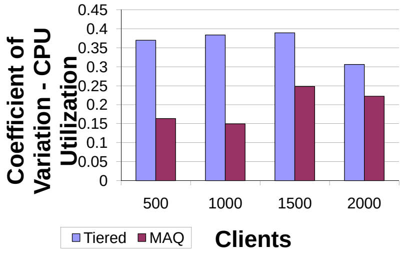


Figure 15: Coefficent of Variation of Node Usage

It can be seen that the CV for the tiered deployment is much higher than the MAQ-PRO deployment. This signifies that the MAQ-PRO deployment uses the processors in a more balanced manner than the tiered deployment reinforcing our claim that MAQ-PRO effectively utilizes resources. The outcome is the ability of MAQ-PRO to handle more incoming load while preventing a single node to become the bottleneck as long as possible.

2. Handling Increasing Number of Clients:

Our MAQ-PRO algorithm also enables increasing the number of clients handled using the same fixed number of machines compared to a tiered architecture. This result can be achieved with a slight variation of Algorithm 2 where the number of nodes are fixed initially to some value. The algorithm terminates as soon as the response time reaches the SLA bound (which means that performance is assured).

Using the result of three nodes obtained in the previous result, we conducted additional experiments. The allocation decisions made by MAQ-PRO are used to place the components on the machines and the number of clients is gradually increased till their response times reach a SLA bound of 1 sec. In comparison, the tiered deployment is also used to host the same number of clients.

Figure 16 shows the response time for both the tiered deployment and the MAQ-PRO deployment. It can be seen that the tiered deployment reaches a response time of 1 sec at around 1,800 clients while the MAQ-PRO deployment reaches a response time of 1 sec at around 2,150 clients. This result shows an improvement of 350 clients or around 20% thereby providing an opportunity for service providers to increase their revenues.

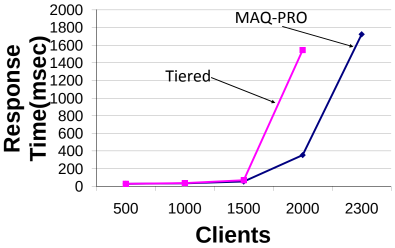


Figure 16: Response Time Comparison

4. RELATED WORK

This section compares MAQ-PRO against related work along two dimensions.

Analytical and Profile based Techniques: A large body of work on analytical techniques to model and estimate the performance of multi-tiered internet applications exists. For example, [11, 19, 21, 23, 24] use closed queuing networks to model multi-tiered internet applications. These efforts typically model an entire tier as a queue. Such models are also usually service-aware, which allows system management decisions involving components and services to be executed.

In contrast, MAQ-PRO models the applications at the granularity of a software component. The finer granularity helps our heuristics to place components onto nodes so that resource wastage is minimized. In addition, load dependent service demands are used to model increased system activity at high utilization levels. MAQ-PRO also presents a method to model blocking effects due to database optimizations [10]. This method ensures that the queuing models remain tractable while simultaneously improving the accuracy of performance predictions.

Stewart et. al. [15] propose a profile-driven performance model for cluster based multi-component online services. They use their model to perform system management and implement component placement across nodes in the cluster. MAQ-PRO complements this work by modeling system activity, multiple processors/cores, and database optimizations. It also uses a formalized queuing model to predict performance.

Application Placement Techniques: Karve et al. [6] and Kimbrel et. al. [7] present a framework for dynamic placement of clustered web applications. Their approach considers multiple resources, some being load-dependent while others are load-independent. An optimization problem is solved which attempts to alter the component placement at run-time when some external event occurs. Components are migrated to respond to external demands.

Carrera et al. [4] design a similar system but they also provide utility functions of applications mapping CPU resource allocation to the performance of an application relative to its objective. Tang et al. [18] propose a placement algorithm which can be used with the MAQ-PRO performance models from Stage 1. Urgaonkar et. al. [22] identify resource needs of application capsules (components) by profiling them.They also propose an algorithm for mapping the application capsules onto the platforms (nodes).

MAQ-PRO differs from these approaches in terms of its workload and performance models, and also in terms of the replication management strategy. MAQ-PRO defines a queuing model and enhances it to consider application- and hardware-specific factors which influence the performance of the applications. The queuing model captures the interference due to multiple components being co-located together. Since MAQ-PRO is a strategizable framework, the placement algorithms in [6, 7, 18, 22] can be plugged in.

None of the prior works above (except [19]) enforces explicit performance bounds. MAQ-PRO maintains performance bounds through the use of SLAs. The placement of the components is thus attempted to maximize capacity while ensuring that the performance remains within specified SLA bounds.

5. CONCLUDING REMARKS

This paper presented the MAQ-PRO process which is a two stage framework comprising techniques to develop profile-based analytical models, and an algorithm for component replication and allocation for multi-tiered, component-based applications. The goal of the MAQ-PRO process is high assurance of performance and service availability to users, while minimizing operating costs and potentially improving revenues to the service provider.

MAQ-PRO advocates a profiling method by which traditional queuing models can be enhanced and made more accurate. The novel ideas include the use of load-dependent service demands of individual services on the processor and correction factor for easily estimating multi-processor activity. MAQ-PRO also provides a component replication and allocation algorithm which makes use of the above analytical model in minimizing the number of resources used and balancing their usage while meeting the target number of clients and their SLA bounds. It is shown that by keeping the resource utilization of each component within a certain threshold such as 30% of CPU time, the resources can be utilized better.

We have used a running example of the RUBiS web portal to discuss the two stages of MAQ-PRO and discussed the steps any capacity planner should undertake when applying MAQ-PRO to their applications. In the context of RUBiS, MAQ-PRO was shown to have saved 25% resources while supporting 20% more load when compared to using traditional modeling techniques all while providing high performance and availability assurances to users.

Our results indicate that the process to enhance traditional queuing models with profiling based measurements helped us to derive more accurate models. Since our approach is profile-based, the empirical results depend upon the software design, business logic, and underlying hardware. Thus the models developed for RUBis may not apply directly to other projects. On the other hand such software behavior is common across many applications and our profiling techniques can be repeated on the concerned platform/projects to measure the required variables, and derive enhanced analytical models.

Our future work will investigate the impact of resource failures and include fault tolerance. The MAQ-PRO data and algorithm is available at `http://www.dre.vanderbilt.edu/~nilabjar/MAQ-PRO`.

6. REFERENCES

[1] C. Amza, A. Ch, A. Cox, S. Elnikety, R. Gil, K. Rajamani, and W. Zwaenepoel. Specification and Implementation of Dynamic Web Site Benchmarks. In *5th IEEE Workshop on Workload Characterization*, pages 3–13, 2002.

[2] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience, NY, USA, 1998.

[3] A. Budhiraja and A. Ghosh. A large deviations approach to asymptotically optimal control of crisscross network in heavy traffic. *Annals of Applied Probability*, 15(3):1887–1935, 2005.

[4] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade. Utility-based placement of dynamic web applications with fairness goals. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 9–16, April 2008.

[5] E. Coffman Jr, M. Garey, and D. Johnson. Approximation algos for bin packing: a survey. 1996.

[6] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *Proceedings of the 15th international conference on World Wide Web*, pages 595–604. ACM New York, NY, USA, 2006.

[7] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic Application Placement Under Service and Memory Constraints. In *Experimental And Efficient Algorithms: 4th International Workshop, WEA 2005, Greece, May 10-13, 2005*, Springer, 2005.

[8] S. Kumar and P. Kumar. Closed Queueing Networks in Heavy Traffic: Fluid Limits and Efficiency. *Stochastic networks: stability and rare events*, 1996.

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

[10] A. G. Nilabja Roy and L. Dowdy. A Novel Capacity Planning Process for Performance Assurance of Multi-Tiered Web Applications. In *Proceedings of MASCOTS 2010*, Miami Beach, FL, USA, Aug. 2010.

[11] G. Pacifici, W. Segmuller, M. Spreitzer, M. Steinder, A. Tantawi, and A. Youssef. Managing the response time for multi-tiered web applications. *IBM TJ Watson Research Center, Yorktown, NY, Tech. Rep. RC23651*, 2005.

[12] E. Pekoz and J. Blanchet. Heavy Traffic Limits Via Brownian Embeddings. *Probability in the Engineering and Informational Sciences*, 20(04):595–598, 2006.

[13] N. Roy, J. S. Kinnebrew, N. Shankaran, G. Biswas, and D. C. Schmidt. Toward Effective Multi-capacity Resource Allocation in Distributed Real-time and Embedded Systems. In *Proceedings of the 11th ISORC)*, Orlando, Florida, May 2008. IEEE.

[14] N. Roy, Y. Xue, A. Gokhale, L. Dowdy, and D. C. Schmidt. A Component Assignment Framework for Improved Capacity and Assured Performance in Web Portals. In *Proceedings of the DOA 2009*, pages 671–689, Nov. 2009.

[15] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2 table of contents*, pages 71–84. USENIX Association Berkeley, CA, USA, 2005.

[16] C. Stewart, K. Shen, A. Iyengar and J. Yin. EntomoModel: Understanding and Avoiding Performance Anomaly Manifestations. In *Proceedings of International Symposium of Modeling, Analysis, and Simulation of Computer Systems*, pages 3–13. IEEE Computer Soc, Los Alamitos, CA, USA, 2010.

[17] R. Suri, S. Sahu, and M. Vernon. Approximate Mean Value Analysis for Closed Queuing Networks with Multiple-Server Stations. In *Proceedings of the 2007 Industrial Engineering Research Conference*. 2007.

[18] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, page 340. ACM, 2007.

[19] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An Analytical Model for Multi-tier Internet Services and its Applications. *SIGMETRICS Perform. Eval. Rev.*, 33(1):291–302, 2005.

[20] B. Urgaonkar, A. Rosenberg, P. Shenoy, and A. Zomaya. Application Placement on a Cluster of Servers. *International Journal of Foundations of Computer Science*, 18(5):1023–1041, 2007.

[21] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. International Conference on*, pages 217–228, 2005.

[22] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in a shared Internet hosting platform. *ACM Transactions on Internet Technologies (TOIT)*, 9(1):1–45, 2009.

[23] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: a capacity planning and anomaly detection tool for enterprise services with live workloads. In *Middleware '07: ACM/IFIP/USENIX 2007 International Conference on Middleware*, pages 244–265, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[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.