A Capacity Planning Process for Performance and Availability Assurance of Multi-Tiered Web Applications

Nilabja Roy, Aniruddha Gokhale and Larry Dowdy
*Dept. of EECS, Vanderbilt University, Nashville, TN 37235, USA
Email: nilabjar@dre.vanderbilt.edu,{a.gokhale,larry.dowdy}@vanderbilt.edu

***Abstract*—**
For service providers of multi-tiered 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 allocate software components on minimum number of resources.

Our results show 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.

***Keywords*-Multi-tier applications, performance estimation, service deployment.**

I. Introduction

Multi-tiered internet-based applications such as web portals (*e.g.*, eBay, Priceline, Amazon and Facebook) provide a variety of services that support a large number of concurrent users. A common requirement across all these multi-tiered applications is providing 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 and honor SLAs is for the service providers to provision a large number of resources. However, such an approach often leads to less than desired utilization of resources, and significantly higher procurement and operational costs for the service provider that impacts revenues.

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.

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). The capacity planning problem requires service providers to effectively deploy and configure their services (which themselves are made up of a collection of interacting software components) on minimum number of system resources including the CPUs, networks, databases, and storage devices, among others.

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 very accurate 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 [1]–[5] exists but these efforts have not accounted for increased system activity, such as page-faults which occurs with increased load, which is a frequent phenomenon. 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, this coarse level of granularity is insufficient in minimizing the number of and efficiently using resources in the context of modern multi-tiered systems that services made up of finer-grained software components.

In our previous preliminary work [8], we have identified key impediments to accurate analytic modeling of various scenarios commonly occurring in most web portals. Through this study, we concluded that since each application has its

own traits, they can be well-understood only with detailed profiling. Moreover, detailed profiling when combined with analytical modeling holds promise in developing more accurate system models, which in turn helps with effective capacity planning, and higher assurance of performance and availability properties.

This paper builds upon the promising directions shown in our preliminary work [8]. It develops and presents a two-stage, design-time, capacity planning process that systematically combines the strengths of analytical modeling, profiling, and allocation heuristics in a novel framework called MAQ-PRO (Modeling and Analysis using Queuing, Placement and Replication Optimizations). The MAQ-PRO process hinges on a component-based structure of multi-tiered applications. This fine level of granularity is justified since applications and services are increasingly becoming component-based. Moreover, it provides significant flexibility in deploying the services.

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 system called Rice University Bidding System (RUBiS) [9]. It is a prototype of an auction site that mimics eBay.

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

II. MAQ-PRO Capacity Planning Process

Our goal for this paper is to provide performance and availability assurances to the users of multi-tiered applications, such as web portals, without unduly affecting revenues for the service provider. Providing these assurances depends primarily[1] on how many resources are provisioned and how they are utilized. Again cost minimization dictates utilizing minimum number of resources. Thus solving the high assurance problem in this case reduces to solving the capacity planning problem for multi-tiered applications.

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 resulting in a two-stage process architecture where information from one stage is seamlessly handed over to the next.

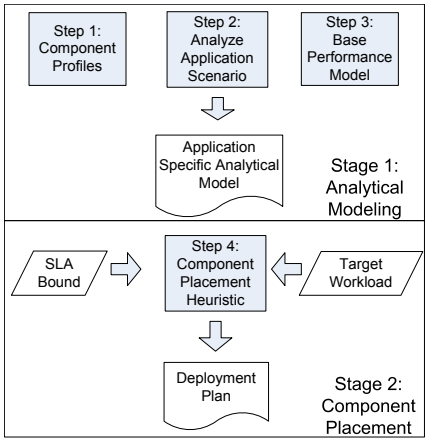


Figure 1. **The MAQ-PRO Process for Capacity Planning**

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 and balanced use of resources, which in turn provides assurances of performance and availability to the users.

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 that emulates behavior of users.

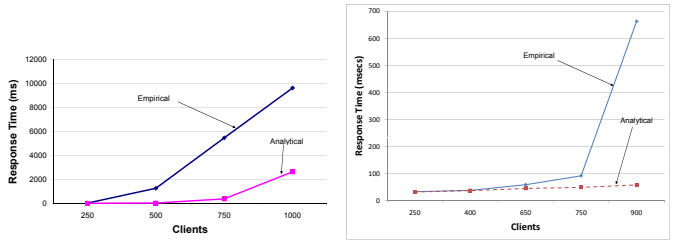
[1]We do not consider network delays in this paper.

A. *Stage 1: Estimating System Performance via Modeling*

Recall from Section I that contemporary analytical techniques are limited in their ability to accurately estimate performance of a given system in the presence of high system activity (*Limitation 1*) and also while operating on multiprocessor/core architectures (*Limitation 2*). Next we describe how Stage 1 of the MAQ-PRO process addresses these limitations in the context of RUBiS. Capacity planners should adopt a similar approach for their application mix.

*1) **Overcoming Limitation 1: Modeling Increased System Activity**:* In our previous work [8] we showed how a queuing model used in related research [1]–[4] does not provide accurate response time estimates when the client population increases. High client population increases the number of threads (*i.e.*, concurrency) and thus increases system activity, which is typically not accounted for in the model. Figure 2a shows this behavior when a single service of RUBiS is run. All our experiments are conducted in ISISLab www.isislab.vanderbilt.edu/. Each machine has two 2.8 GHz Xeon CPUs, 1GB of ram and 40GB HDD.

Similar behavior is also seen when multiple services run, as shown in Figure 2b. Here we reproduce only the service "SearchByCategory" which has higher response times. The other services also incur similar estimation errors.



(a) Single Service (b) Multiple Service

Figure 2. **Comparison of Analytical vs Empirical Data**

We overcome this limitation by treating each device, *e.g.*, CPU and disk, where system activity is manifested – as a load-dependent device [10]. We use the term *service demand* to denote the amount of time each transaction takes to execute on a resource. Thus, a load dependent device is one whose service demand varies with load. Service demands can be measured using the service demand law [10]. 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. The load-dependent service demand 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 service demand.

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 service demand for each client size.

The load-dependent service demand for the RUBiS `SearchItemsByRegion` service is shown in Figure 3. The figure illustrates that the service demand varies with client population. The corresponding processor utilization are shown in parenthesis on the X axis. The dashed line plots the number of context switches that occur per second for different processor utilizations. Context switching is a measure of the amount of system activity.

As seen in Figure 3, the 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 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.

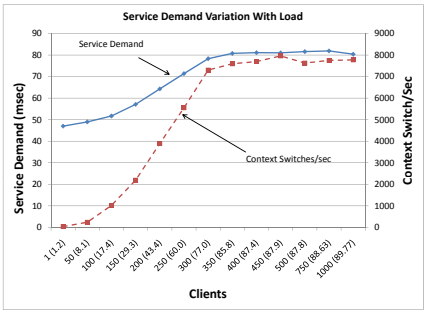


Figure 3. **Service Demand of SearchItemsByRegion**

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

$$SD_{sr}(U) = \begin{cases} 48 & \text{for } U < 8 \\ 0.4264 \times U + 45.1062 & \text{for } 8 <= U <= 85 \\ 81.62 & \text{for } U > 85 \end{cases} \quad (1)$$

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

$$SD_{sc}(U) = \begin{cases} 28 & \text{for } U \leq 5 \\ 0.0457 \times U + 24.94 & \text{for } 5 <= U <= 84 \\ 52.06 & \text{for } N \geq 84 \end{cases} \quad (2)$$

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

*2) **Overcoming Limitation 2: Modeling Multiprocessor Effects**:* Due to the increasing availability and use of multi-processors and multi-cores for multi-tiered applications, such as web portals, existing closed queuing network models [1], [4], [11] must now incorporate support for multiple servers. Although existing closed queuing networks can be solved efficiently using the mean value analysis (MVA) algorithm, accounting for multiple-server models requires 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.

To address this challenge, we leverage recent results [12] in which a simple approximate method is presented that extends MVA to analyze multiple-servers. In this related work, the authors introduce the notion of a *correction factor*, which estimates the waiting time. When a transaction is executed on multi-processor 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:

$$R(N) = SD(N) + c \times SD(N) \times n \quad (3)$$

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.

The value of the service demand SD can be found using the profile-based curve fitting approach explained in Section II-A1. 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 3 is the correction factor, c, which can be obtained by solving the equation.

Computing the correction factor, however, is not an easy task since it may be dependent on a number of factors, such as the domain of the operation, and the service time characteristics for the underlying hardware. Therefore, the correction factor will vary with each scenario. 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.

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, $SD(N)$, were obtained from Equations 1 and 2. The correction factor was then computed using Equation 3, which is presented in Table I for two different services in RUBiS for a 4 processor machine.

Table I 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.

Based on the earlier data, we surmised that the correction factor may vary with the number of processors in the machine. To support our believe, we configured the machine to use different number of processors and repeated the experiment with 1 and 2 processors, respectively. Figure 4 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.

It is clear that the value of CI has a very high value with less load but slowly converges to a steady value at high load. The steady value seems to be equal 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 Equestion 3. 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.

Service Name	Clients	Service Demand (msec)	AvgWaiting	Response Time	Corr. Factor	CI
SearchItemsByReg	100	51.71	2.00	54	0.022	45.16
	150	57.12	2.00	62	0.043	23.40
	200	64.29	3.00	77	0.066	15.17
	250	71.4	5.00	103	0.089	11.29
	300	78.3	10.00	222	0.183	5.45
	350	80.78	40.00	909	0.256	3.90
	400	81.12	86.00	1968	0.27	3.69
	500	81.62	185	4232	0.275	3.64
SearchItemsByCat	100	51	2	54	0.029	34.00
	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 I
Correction Factors for Various Services

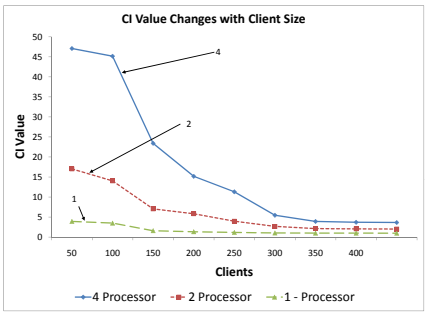


Figure 4. **Inverse of Correction Factor (CI)**

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

$$CI(U) = \begin{cases} -0.5632 \times U + 38.75 & \text{for } U <= 58 \\ -0.1434 \times U + 15.71 & \text{for } 58 < U < 85 \\ 3.69 & \text{for } U \geq 85 \end{cases} \quad (4)$$

Equation 4 is then used from within MVA algorithm to compute the response time in each iteration.

*3) **Modifying Mean Value Analysis Algorithm**:* We develop a multi-class closed queuing model for RUBiS as shown in Figure 5 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 is modeled as a job class.

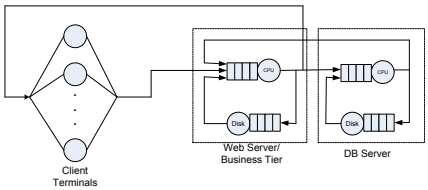


Figure 5. **Queuing Model of RUBiS Scenario**

An approximate MVA algorithm 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 [10]. 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 operates in an iterative manner increasing the number of clients from a starting value of 1 to the final value N. In each iteration, it 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.

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

1 **begin**
2 // Run initial MVA with lowest service demand
3 **while** $Error > \epsilon$ **do**
4 // Initialization ....
5 **for** $r \leftarrow 1$ **to** R **do**
6 **for** $i \leftarrow 1$ **to** K **do**
7 $\mathbf{D_{i,r}} = \mathbf{SD_{i,r}(U_r)}$ // Call function for Service Demand with device utilization as parameter
8 $R_{i,r} = D_{i,r} \times (1 + \mathbf{CI(U_r)} \times n_r)$
9 $X_r = \frac{N_r}{z_r + \sum_{i=1}^{K} R_{i,r}}$
10 // Error = Maximum Difference in Utilization between successive iterations

The boldface parts shown are the places where the original MVA algorithm is modified to include the functions for service demand and correction factor. The function $SD_{i,r}$ represents the service demand function for i^{th} job class in the r^{th} device while function $CI(U_r)$ is the function 4. Both these need a 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 service demand for each service as given by Equations 1, 2 and the value of CI equal to the number of processors in the system.

B. Stage 2: Minimizing and Balancing 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 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 6 from our earlier work [11] in which processor utilizations of two servers in RUBiS are shown. In the machine `DB_SRV`, only one component called component `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`

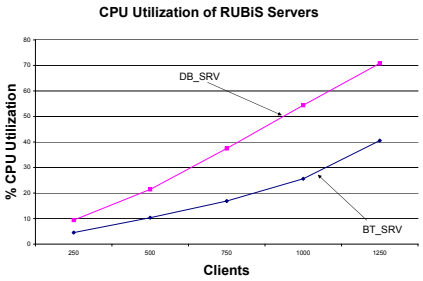


Figure 6. **CPU Utilization**

is loaded only upto 40%. Thus, there is an imbalance in resource usage.

Resource allocation algorithms developed in prior research [6], [7], [11] 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, 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 an allocation will make 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 control the number of resources required and utilized by the component. 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 [13].

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:

$$\#ofmachines = \lceil Ld/m \rceil <= OPT, \qquad (5)$$

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 [14]. 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 [13]. Thus, popular heuristics like first-fit, best-fit or worst-fit [14] 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 [14].

Our previous work [15] 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 II. 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.

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 II
Success Rate of Heuristics on Solvable Problems:Courtesy [15]

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 5.

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.

Algorithm 2: Replication & Allocation

```
   begin
       // Initially, use 2 machines in a tiered deployment
1
2      // All business logic components in first machine
3
4      // Database in second machine, Default Deployment Plan DP
5
6      P = 2 // Initially 2 machines
7      N = init_clients
8      (RT,SU,U) = MVA (DP, N) // Compute Initial Component
         Utilizations
9      (DP) = Placement (SU, P) // Find a placement of the
         components
10
11     while N < Target do
12         (RT,SU,U) = MVA (DP, N)
13         if ∃i : SU_i > 30 then
14             Replicate (i); // Create New instance of Component i
15
16             // Place new component on same machine as i
17             (RT, SU, U) = MVA (DP, N) // Calculate new
                 response time
18             (DP) = Placement (SU, P) // Update Deployment
                 Plan
19         if ∃i : RT_i > RT_SLA then
20             // add new machine
21             P = P + 1
22             (DP) = Placement (SU, P) // find new placement
23         N += incr // Increase Clients for next iteration
```

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 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.

III. Evaluating the MAQ-PRO Framework

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.

A. *Stage I Model Validation*

Our objective in validating the enhanced model resulting from Stage I seeks to understand how close the estimated response time from our enhanced models are to that of the empirically measured values. The model is used to predict the performance of the application when multiple service types are run. 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.

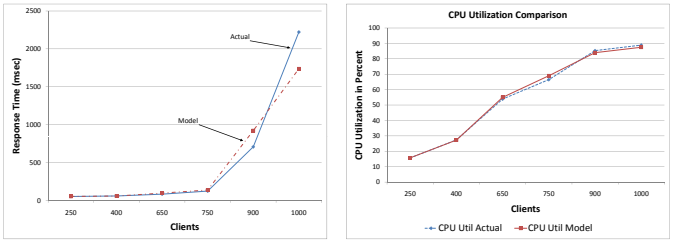
Figure 7a shows the response time estimated by our model for one service, "SearchByRegion". The estimation of the other services are also similar. It can be seen that our enhanced model is in close agreement with the empirical measurements till the number of clients equal to 900. Beyond that number, the error in our model increases slightly but still is close to the actual result.

Figure 7b 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.

B. *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:



(a) SearchByRegion Response Time (b) CPU Utilization

Figure 7. **Validation of Model Estimation**

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 8 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 give good results due to the robustness of closed non-product-form queuing networks. By robustness, it is meant that a major change in system parameters will bring about tolerable changes in computed parameters [16].

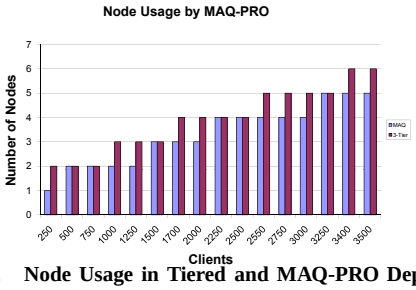


Figure 8. **Node Usage in Tiered and MAQ-PRO Deployments**

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.

		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 III
Response Time and Utilization

Table III shows the response times and the utilizations of 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 9 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.

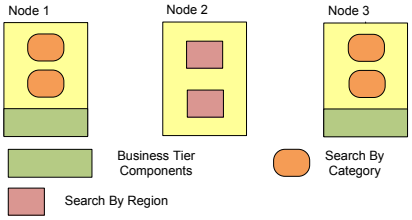


Figure 9. **Allocation of Components for 2,000 Client**

Figure 10 presents the coefficient of variance (CV) of the CPU usages for the three machines used in this experiment. 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.

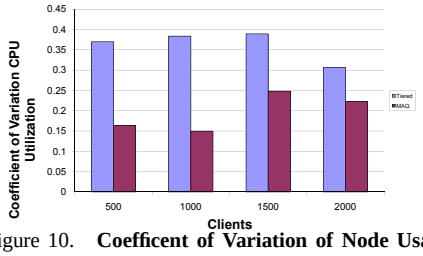


Figure 10. **Coefficent of Variation of Node Usage**

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 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 11 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.

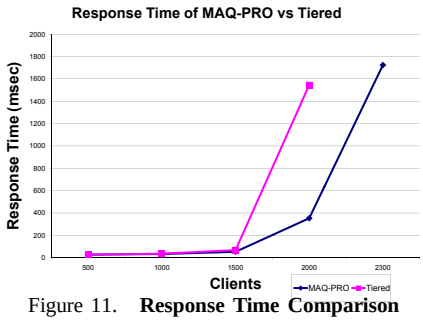


Figure 11. **Response Time Comparison**

IV. 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, [1]–[4], [17] 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 [8]. This method ensures that the queuing models remain tractable while simultaneously improving the accuracy of performance predictions.

Stewart et. al. [5] 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. [18] 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. [19] propose a placement algorithm. MAQ-PRO models from Stage 1 can be used with their algorithm. Urgaonkar et. al. [20] 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], [19], [20] can be plugged in.

None of the prior works above (except [1]) 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.

V. 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.

REFERENCES

[1] 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.*, vol. 33, no. 1, pp. 291–302, 2005.

[2] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, 2005, pp. 217–228.

[3] 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: Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2007, pp. 244–265.

[4] Q. Zhang, L. Cherkasova, N. Mi, and E. Smirni, "A regression-based analytic model for capacity planning of multi-tier applications," *Cluster Computing*, vol. 11, no. 3, pp. 197–211, 2008.

[5] 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*. USENIX Association Berkeley, CA, USA, 2005, pp. 71–84.

[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*. ACM New York, NY, USA, 2006, pp. 595–604.

[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, Santorini Island, Greece, May 10-13, 2005: Proceedings*. Springer, 2005, p. 391.

[8] A. G. Nilabja Roy and L. Dowdy, "A Novel Capacity Planning Process for Performance Assurance of Multi-Tiered Web Applications," in *To Appear in the Poster Proceedings of the 18th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '10)*. Miami Beach, FL, USA: IEEE, Aug. 2010.

[9] 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*, 2002, pp. 3–13.

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

[11] 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 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA'09)*, Nov. 2009, pp. 671–689.

[12] 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*. Citeseer, 2007.

[13] B. Urgaonkar, A. Rosenberg, P. Shenoy, and A. Zomaya, "Application Placement on a Cluster of Servers," *International Journal of Foundations of Computer Science*, vol. 18, no. 5, pp. 1023–1041, 2007.

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

[15] 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 International Symposium on Object/Component/Service-oriented Real-time Distributed Computing)*. Orlando, Florida: IEEE, May 2008.

[16] 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 New York, NY, USA, 1998.

[17] 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.

[18] 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*, April 2008, pp. 9–16.

[19] 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*. ACM, 2007, p. 340.

[20] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in a shared Internet hosting platform," *ACM Transactions on Internet Technologies (TOIT)*, vol. 9, no. 1, pp. 1–45, 2009.