A Component Assignment Framework for Improved Capacity and Assured Performance in Web Portals

Nilabja Roy, Yuan Xue, Aniruddha Gokhale, Larry Dowdy and Douglas C. Schmidt

Electrical and Computer Science Department Vanderbilt University

Abstract. Web portals hosting large-scale internet applications have become popular due to the variety of services they provide to their users. These portals are developed using component technologies. Important design challenges for developers of web portals involve (1) determining the component placement that maximizes the number of users/requests (capacity) without increasing hardware resources and (2) maintaining the performance within certain bounds given by service level agreements (SLAs). The multitude of behavioral patterns presented by users makes it hard to identify the incoming workloads.
This paper makes three contributions to the design and evaluation of web portals that address these design challenges. First it introduces an algorithmic framework that combines bin-packing and modeling-based queuing theory to place components onto hardware nodes. This capability is realized by the Component Assignment Framework for multi-tiered internet applications (CAFe). Second, it develops a component-aware queuing model to predict web portal performance. Third, it provides extensive experimental evaluation using the Rice University Bidding System (RUBiS). The results indicate that CAFe can identify opportunities to increase web portal capacity by 25% for a constant amount of hardware resources and typical web application and user workloads.

1 Introduction

Emerging trends and challenges. Popular internet portals, such as eBay and Amazon, are growing at a fast pace. These portals provide many services to clients, including casual browsing, detailed reviews, messaging between users, buyer and seller reviews, and online buying/bidding. Customers visiting these web portals perform various types of behaviors, such as casual browsing, bidding, buying, and/or messaging. The broader the variety of services and user behavior a web portal supports, the harder it is to analyze and predict performance.

Performance of web portals can be quantified by average response time or throughput, which are functions of incoming load. In turn, the load generally corresponds to the arrival rate of user sessions or the concurrent number of clients handled by the portal. As the number of clients increases, performance can degrade with respect to customer's response time.

Service level agreements (SLAs) could be provided to bound the performance, such as an upper bound on response time. In such a case, a web portal needs to meet performance within the bounds given by the SLA. The capacity of such a web portal is

defined as the maximum number of concurrent user sessions or the maximum request arrival rate handled by applications with performance within the SLA bound.

The goal of portal designers is to deploy the given web portal to maximize its capacity, *i.e.*, maximize the number of users in the system while maintaining the average response time below the SLA limit. In turn, this goal helps maximize the revenue of the portal since more users can be accommodated. Portal capacity is generally proportional to the hardware, *i.e.*, more/better hardware means more/better capacity. Web portal designers are constrained by system procurement and operational costs, however, which yields the following questions: (1) for a fixed set of hardware, can web portals serve more clients and is the current hardware utilized in the optimal way?

To answer these questions, it is important to understand the way contemporary web portals are implemented. Component-oriented development (such as J2EE or .NET) is generally used to develop web portals by using the "divide and conquer" method. Each basic functionality (such as a business logic or a database query) is wrapped within a component, such as a Java class. Several components are then composed together to implement a single service, such as placing bids in an auction site or booking air tickets in a travel site.

A service is realized by deploying the assembly of components on the hardware resources. Maximizing the utilization of resources and meeting client SLA requirements can be achieved by intelligently placing the components on the resources. For example, colocating a CPU-intensive component with a disk-intensive component may yield better performance than two CPU-intensive components together.

Application placement in clustered servers has been studied extensively [1–6]. For example, [4–6] estimate resource usage of components and place components onto nodes by limiting their sum of resource usage within available node resource. They do not check response time of applications. On the other hand, [1] checks the response time while placing components. Their model is based on linear fitting and the effect of colocating components is estimated approximately. Other work [2, 3] checks response time while provisioning resources using queuing theory.

In most related work, however, the functional granularity is modeled at the tier level, not at the component level. The downside of using the coarser granularity is that resources can remain unutilized in nodes where components could be configured, but tiers may not. If the models are component aware, available resources in the nodes can be better utilized by proper placement. Algorithms become more complicated, however, since the solution space is increased.

Solution approach → Intelligent component placing to maximize capacity . This paper presents the *Component Assignment Framework for multi-tiered internet applications* (CAFe), which is an algorithmic framework for increasing capacity of a web portal for a fixed set of hardware resources by leveraging the component-aware design of contemporary web portals. The goal of CAFe is to create a deployment plan that maximizes the capacity of the web portal so their performance remains within SLA bounds. It consists of a mechanism to predict the application performance coupled with an algorithm to assign the components onto the nodes.

CAFe complements related work by (1) introducing the concept of a performance bound through a SLA that acts as an additional constraint on the placement prob-

lem, which requires estimating and evaluating application performance against the SLA bound at every step, (2) creating service- and component-aware queuing models to estimate component resource requirements that predict overall service performance, (3) devising an algorithmic framework that combines a queuing model with a bin-packing algorithm to iteratively compute component placement while maximizing capacity of the application, and (4) showing how to balance resource usage across nodes in the network to deliver better performance.

Paper organization. The rest of the paper is organized as follows: Section 2 discusses an example to motivate intelligent component placement and formulate the problem CAFe is solving; Section 3 examines CAFe's algorithmic framework in detail; Section 4 empirically evaluates the performance of CAFe; Section 5 compares CAFe with related work; and Section 6 presents concluding remarks.

2 Motivating the Need for CAFe

This section presents an example to motivate intelligent component placement and formulate the problem CAFe is solving.

2.1 Motivating Example

The motivating example is modified from [7] to show how intelligently mapping the components of a web portal to available hardware nodes increases performance by several factors. The portal is an online bidding system (similar to eBay) that provides multiple services, such as browsing, bidding, and creating auctions. The portal is structured as a 3-Tier application with web-server, business logic, and database tiers.

Each service in the web portal consists of three components, one in each tier of the application. For example, the functionality of creating auctions is implemented by a service that is composed of three components: (1) a web service component that handles incoming requests, (2) a business tier component that implements application logic, and (3) a database component to make database calls. The web portal is deployed using a default deployment strategy with each tier placed on a single node. The portal performance is analyzed using an analytical model presented in [7].

Server	CPU %	Disk %
Web Server	0.510	0.568
BT Node	0.727	0.364
DB Server	0.390	0.997

(a) Original Deployment

Server	CPU %	Disk %
Web Server	0.510	0.568
BT Node	0.822	0.554
DB Server	0.294	0.806

(b) After Reallocation

Table 1: **The Utilization of resources**

The response times are given in Table 2, 2^{nd} column. Most services have an unduly high response time, *e.g.*, Place Bid has a response time of 30 secs, which is unacceptable in an online bidding system. The CPU and disk utilizations of each node is given in Table 1a. Table 1a shows that the DB Server is disproportionately loaded. In particular, the DB Server disk is overloaded, whereas the Business Tier (BT) Server disk is

underloaded. One solution would be to move some components from the DB Server to the Business Tier Server so disk usage is more uniform across the nodes.

The database component for the "Login" service is chosen at random to move to the BT node. To analyze the new deployment, the analytic model is changed and the new utilization of the resources are given in Tables 1b, which shows that the resources are more evenly utilized, *e.g.*, the DB_Server disk is now only 80% utilized. The corresponding response times are given in the Table 2, $3^{rd}, column$. The percent decrease

Services	Response Time	New Response Time	% Decrease
Home	0.086	0.086	0.00
Search	12.046	0.423	96.49
View Bids	6.420	0.530	91.78
Login	17.230	0.586	96.60
Create Auction	27.470	0.840	96.95
Place Bid	30.800	0.760	97.53

Table 2: The Response Times of the Services Before and After Reallocation

in the response times are given in the right-most column. These results show that the response times are reduced significantly.

The analysis clearly shows that the response time of all services can be improved significantly by placing the components properly. Proper placement can be achieved via two approaches. First, component-aware analytical models can be used to evaluate the impact of co-located components. Traditional models have looked at performance from the granularity of a tier [2] or have overlooked the effect of co-locating components [1]. Second, resource utilization can be balanced between the various nodes to ensure the load on any one resource does not reach 100%. This approach can be aided by analyzing performance at the component level (since components help utilize available resources more effectively) and using a placement routine that maps components to nodes so that the resources are utilized uniformly and none utilize 100% of any resource. Section 3 shows how CAFe combines both approaches to develop a component mapping solution that improves web portal capacity.

2.2 Problem Formulation and Requirements

As discussed in Section 1, the problem CAFe addresses involves maximizing the capacity (user requests or user sessions) of a web portal for given hardware resources, while ensuring that the application performance remains within SLA bounds. This problem can be stated formally as follows: The problem domain consists of the set of n components C, $\{C_1, C_2, ...C_n\}$, the set of m nodes P $\{P_1, P_2 ... P_m\}$, and the set of k services $\{S_1, S_2 ... S_k\}$. Each component has Service Demand D $\{D_1, D_2 ... D_n\}$. Each service has response time RT $\{RT_1, RT_2 ... RT_k\}$. The capacity of the application is denoted by either the arrival rate, λ for each service $\{\lambda_1 .....\lambda_k\}$ or the concurrent number of customers M $\{M_1, M_2, ...M_k\}$. $SU_{i,r}$ gives the utilization of resource r by component i. The SLA gives an upper bound on the response times of each service $k\{RT_{sla,1} ... RT_{sla,k}\}$.

CAFe must therefore provide a solution that places the n components in C to the m nodes P such that the capacity (either λ or M) is maximized while the response time is within the SLA limit $RT < RT_s la$. To achieve this solution, CAFe must meet the following requirements:

Place components onto nodes to balance resource consumption in polynomial time. Application components must be placed onto the available hardware nodes such that application capacity is maximized while the performance is within the upper bound set by a SLA. Since this is an NP-Hard problem [8] it is important to find out efficient heuristics that can find good approximate solutions. Section 3.1 describes how CAFe uses an efficient heuristic to place the components onto the available nodes and also ensuring that the resource utilization is balanced.

Estimate component requirement and application performance for various placement and workload. To place the components in each node, the resource requirement of each component is required. Moreover, for each placement strategy, the application performance must be compared with the SLA bound. Both vary with workload and particular placement. We therefore need a workload- and component-aware way of component resource requirement and application performance estimation. Section 3.2 describes how CAFe develops an analytical model to estimate component resource requirement and application performance.

Co-ordinate placement routine and performance modeling to maximize capacity. For each particular placement, the application performance need to be estimated to check if it is within the SLA limit. Conversely, performance estimation can only be done when a particular placement is given. Placement and performance estimation must therefore work closely to solve the overall problem. Section 3.3 describes how CAFe designs a algorithmic framework to co-ordinate the actions of a placement routine and an analytical model in a seamless fashion.

3 CAFe: A Component Assignment Framework for Multi-Tiered Web Portals

This section discusses the design of CAFe and how it addresses the problem and requirements presented in Section 2.2. CAFe consists of two components: the placement algorithm and analytical model, as shown in Figure 1. The input to CAFe includes the set of application components and their inter-dependancy, as shown by the box on the left in Figure 1 and the set of available hardware nodes shown on the right. The output from CAFe is a deployment plan containing the mapping of application components to nodes. This mapping will attempt to maximize the application capacity. The rest of this section describes each element in CAFe.

3.1 Allocation Routine

As mentioned in Section 2.2, there is a need to develop efficient heuristics to place components onto nodes. Placing components to hardware nodes can be mapped as a bin-packing problem [9], which is NP-Hard in the general case (and which is what our scenarios present). Existing bin-packing heuristics (such as first-fit and best-fit) can be used to find a placement that is near optimal.

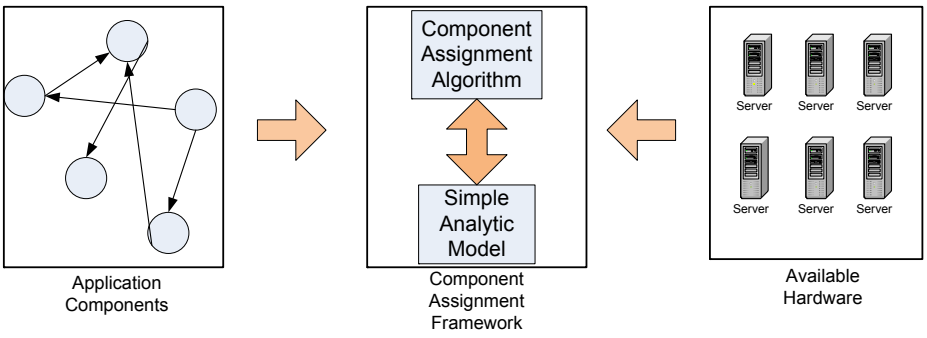


Fig. 1: The CAFe Component Assignment Framework Architecture

The motivating example in Section 2 describes the intuition behind the allocation routine, which is that performance increases by balancing the resource utilization between the various nodes and not allowing the load of any resource to reach 100%. We use worst-fit bin packing since it allocates items to bins by balancing the overall usage of each bin.

Algorithm 1 gives the overall allocation routine, which is a wrapper around the

Algorithm 1: **Allocate**

C: Set of components
foreach *L:Set of Components that should remain local* **do**
  D← Sum the Service Demands of all components in L
  Replace all components in L with D in C
end
$DP = worst\_fit\_bin\_packing(C, P)$

worst-fit Algorithm 2. Algorithm 1 groups together components that are constrained to remain co-located in one machine. For example, they could be the database components that update the tables and need to be connected to the master instance of a database and hence msut be allocated to a single node. Algorithm 1 sums the Service Demands of the components that must be collocated and then replaces them by a hypothetical single component. A call to the $worst\_fit$ routine is made at the end of algorithm 1.

The worst-fit routine is given in Algorithm 2. The components are first ordered according to their resource requirements (Line 2). The algorithm then runs in an iterative fashion. At each step an item is allocated to a bin. All bins are inspected and the least utilized bin is selected.

3.2 Component- and Service-aware Analytical Modeling

Section 2.2 also discusses the need for performance estimation to place components. This estimation process involves (1) predicting the resource requirements of each component for certain application loads, (2) predicting the response time of each service for

Algorithm 2: **Worst-Fit Bin Packing**

```
   begin
     Order_Components(C) // Order the Components by resource requirement
1    foreach C_i ∈ C, 1 ≤ i ≤|C| do
2        // For all components
3      P_k ← The node with the maximum slack
4      Place C_i on to P_k
5    end
   end
```

a particular placement, and (3) computing the overall resource utilization of each node. As application components move among the various nodes, the performance of each service in the application will vary. Performance also depends upon the components that are collocated.

CAFe provides a queuing model that provides average case performance analysis of a system. It also models the interaction of collocated multiple components by modeling the queuing delay for resource contention. CAFe uses this information to transform a deployment plan produced by Algorithm 1 into a multiple class queuing model that maps each service as a class in the model. The components of a single class that are placed in the same node are considered as a single entity. Their Service Demands are summed together. After the model outputs the results, CAFe maps the performance parameters of each class onto the services.

Figure 2 shows the process of creating a model of the application. The input to such

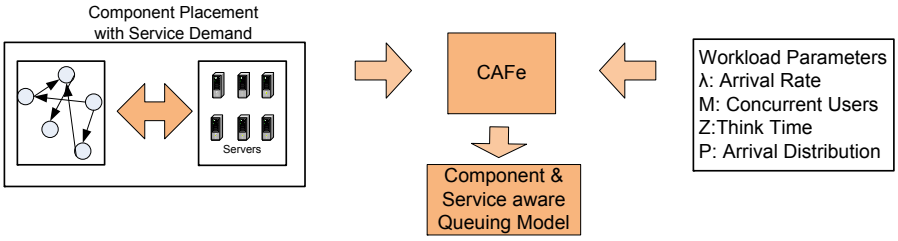


Fig. 2: Create Models of Application

a process consists of the component placement map (mapping of application components to nodes) along with their Service Demands and the workload parameters, such as arrival rate of transactions and number of concurrent user sessions.

Depending upon the workload characteristics, a closed or an open model of the application is constructed. An open model assumes a continuous flow of incoming requests with a given average inter-arrival time between clients. A closed model assumes a fixed number of user sessions in steady state. The sessions are interactive and users make requests, then think for some time, and then make a subsequent request. If an application consists of independent requests arriving and being processed, it can be

modeled as an open model. If there are inter-dependent sequences of requests coming from a single user, however, it must be modeled using a closed model.

These analytical model can be solved using standard procedures. In CAFe, the Mean Value Analysis (MVA) algorithm [7] is used to solve closed models, while an algorithm based on the birth-death system is used to solve open models [7]. The solution to the analytical model provides the response times of the various services and also the utilization of the resources such as processor or disk usage in the various nodes.

3.3 Algorithmic Framework to Co-ordinate Placement and Performance Estimation

Section 2.2 also discusses the need for close co-ordination between placing the components and performance estimation. To meet this requirement CAFe provides a framework that standardizes the overall algorithm and defines a standard interface for communication between the placement and performance estimation. This framework also allows the configuration of other placement algorithms, such as integer programming and different analytical models like models based on worst case estimation. Different algorithms or analytical models can be configured in/out to produce results pertaining to the specific application domain or scenario.

CAFe uses an analytical model of the application and a placement routine to determine a mapping of the application components onto the available hardware nodes. CAFe attempts to maximize the capacity of the web portal, while ensuring that the response time of the requests is within the SLA bounds.

Algorithm 3 describes the component assignment framework. The algorithm alternates between invoking the placement algorithm and an evaluation using the analytic model. In each step, it increases the number of clients in the system by a fixed amount and rearranges the components across the nodes to minimize the difference in resource utilization. It then verifies that the response time is below the SLA limit using the analytical model and iterates on this strategy until the response time exceeds the SLA limit.

CAFe takes as input the details (such as the Service Demands of each component on each resource) of the N components to deploy. Service Demand is the amount of resource time taken by one transaction without including the queuing delay. For example, a single Login request takes 0.004 seconds of processor time in the database server. The Service Demand for Login on the database CPU then takes 0.004 seconds. As output, the framework provides a deployment plan(DP, estimated response (RT) time and total utilization (U) of each resource.

The initial capacity is an input ($Init\_Cap$, Line 2). The value of M is set equal to $Init\_Cap$. This capacity is an arrival rate for an open model or "number of users" for a closed model. The capacity (M) is increased in each step by an incremental step *incr* (Line 9) which also can be parameterized ($Incr$). At each iteration, the response time of all services is compared with the SLA provided upper bound (inner while loop at Line 4).

Inside the inner loop, the framework makes a call to the Allocate module (Line 6), which maps the components to the nodes. This mapping is then presented to the Model (Line 7) along with the Service Demand of each component. The Model computes the

Algorithm 3: Component Assignment Framework

Input:
$C \leftarrow$set of N components to be deployed,
$D \leftarrow$set of Service Demands for all components, $D_{i,r} \leftarrow$Service Demand of component i on the device r
$P \leftarrow$set of K available nodes
RT_{sla} set of response time values for each service as specified by the SLA
Output:
Deployment plan $DP \leftarrow$ set of tuples mapping a component to a node,
M: Total Number of concurrent clients
$RT \leftarrow$set of response times for all components
RT_i: Total response time of service i
U_r: Total Utilization of each resource r in the nodes
$SU_{i,r}$: Utilization of resource r by component i
$SU \leftarrow$ set of resource utilization of all components
$Incr$: Incremental capacity at each step
$Init_Cap$:Initial Capacity

```
1  begin
2    Intially, DP = {}, M = Init_Cap, SU = U, RT_i = Σ_r D_{i,r}, incr = Incr
3    while incr > 10 do
4      while ∄i : RT_i > RT_{sla,i} do
5            // Check if any service RT is greater than SLA bound
6        DP = Allocate(SU, P) // Call Placement routine to get a placement
7        (RT, SU, U) = Model(M, D, DP)// Call model to estimate performance for current placement
8        last_M ← M // Save the previous capacity
9        M ← M + incr // Increment the capacity for the next iteration
10     end
11     // At least one service has Response Time greater than SLA bound for current capacity
12     M ← last_M // Rollback to previous iteration's capacity
13     incr ← incr/2 // Decrease incr by half
14     M ← M + incr // Now Increase capacity and repeat
15   end
16   // while(incr > 10)
17 end
```

estimated response time of each service and the utilization of each resource by each component. It also outputs the total utilization of each resource.

The inner loop of Algorithm 3 exits when response time of any service exceeds the SLA provided upper bound (*i.e.*, M reaches maximum capacity), at which point *incr* is set to a lower value (one-half) and the algorithm continues from the previous value of M (Line 12). If the inner loop exits again, the value of *incr* is lowered further (Line 13). The algorithm ends when the value of *incr* is less than 10.

The output of the algorithm is that value of M, which yields the highest capacity possible and also a deployment plan (DP) that maps the application components onto the nodes. Though not provably optimal, the algorithm is a reasonable approximation. An optimal algorithm would require an integer programming routine [10] to obtain the mapping of the components to the nodes. Such an implementation would be NP-Hard, however, and thus not be feasible for large applications. CAFe therefore uses an intuitive heuristic based on the popular worst-fit bin packing algorithm [9].

4 Experimental Evaluation

4.1 Rice University Bidding System

This section describes our experimental evaluation of CAFe, which used the Java servlets version of the Rice University Bidding System (RUBiS) [11] to evaluate its effectiveness. RUBiS is a prototype of an auction site modeled after ebay that has the features

of an online web portal studied in this paper. It provides three types of user sessions (visitor, buyer, and seller) and a client-browser emulator that emulates users behavior.

A RUBiS session is a sequence of interactions for the same customer. For each customer session, the client emulator opens a persistent HTTP connection to the Web server and closes it at the end of the session. Each emulated client waits for a certain think time before initiating the next interaction. The next interaction is determined by a state transition matrix that specifies the probability to go from one interaction to another one. The load on the site is varied by altering the number of clients.

CAFe requires an analytical model of the application. Once the model is constructed and validated, it can be used in CAFe to find the appropriate component placement. The steps required to build the model are (1) compute Service Demand for each service provided for each customer type such as visitor or buyer and (2) build a customer behavior modeling graph of user interactions and calculate the percentage of requests for each service. For our experiments, a workload representing a set of visitor clients were chosen, so the workload consists of browsing by the users and is thus composed of read-only interactions. The components for each service in RUBiS is given in Table 3.

Services	Home		Browse		Browse_Cat		Browse_Reg		Br_Cat_Reg	
Naming	BT_H	_	BT_B	_	BT_BC	DB_BC	BT_BR	DB_BR	BT_BCR	DB_BCR

Services	Srch_It_Cat		Srch_It_Reg		View_Items		Vu_Usr_Info		Vu_Bid_Hst	
Naming	BT_SC	DB_SC	BT_SR	DB_SR	BT_VI	DB_VI	BT_VU	DB_VU	BT_BH	DB_BH

Table 3: Component Names for Each Service

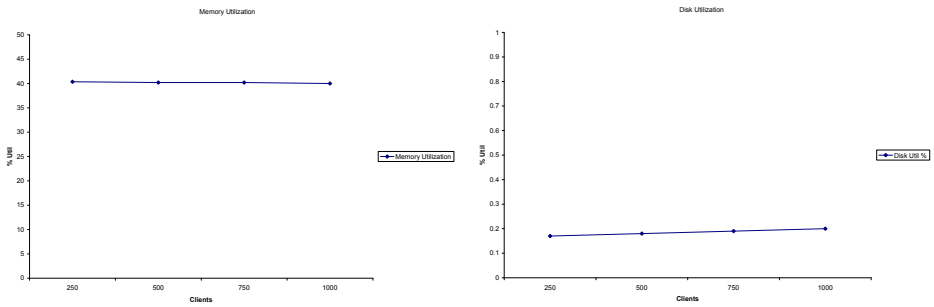
The RUBiS benchmark was installed and run on the ISISLab testbed (`www.isislab.vanderbilt.edu`) at Vanderbilt University using 3 nodes. One for the client emulators, one for the "Business Tier" and the other for "Database Tier". Each node has 2.8 GHz Intel Xeon processor, 1GB of ram, and 40GB HDD running Fedora Core 8.

4.2 Computing Service Demand

The Service Demand of each of the components must be captured to build an analytical model of the application. The RUBiS benchmark was run with increasing clients and its effect on various CPU, memory, and disk were noted. The memory and disk usages are shown in Figures 3a and 3b. Disk usage is low ($\sim$0.2%) and memory usage was $\sim$40%. Moreover, these utilizations remained steady even as the number of clients are increased. Conversely, CPU usages increased as number of clients grew (the Actual line in the Figure 5a).

The results in Figures 3a and 3b show that CPU is the bottleneck device. The Service Demands were computed for the CPU and the disk. Since memory was not used fully, it is not a contentious resource and will not be used in the analytical model. Moreover, the CAFe placement routine ignores disk usage since it remains steady and is much less than CPU usage. The CAFe placement routine thus only uses one resource (CPU) to come up with the placement.

RUBiS simplifies the calculation of Service Demand. It includes a client-browser emulator for a single client and makes requests on one service at a time. During the experiment, the processor, disk and memory usages were captured. After the experiment finished we used the Service Demand law [7] to calculate the Service Demand for that



(a) Memory Utilization (b) Disk Utilization

Fig. 3: **The Utilization of Memory and Disk for RUBiS Benchmark**

service. In some services (such as “Search Items in Categories”) the Service Demand is load dependent. For such services the number of clients was increased and the Service Demands were measured appropriately.

The Service Demands of CPU for all the services measured in such a way are given in Table 4. Each service in RUBiS is composed of multiple components, with a compo-

Service	Business Tier Component(secs)	DB Server Component(secs)	Description
home	0.002	0	Home Page
browse	0.002	0.0	Browse Main Page
browse_cat	0.0025	0.0005	Browse Categories
browse_reg	0.0025	0.0005	Browse Regions
br_cat_reg	0.003	0.0007	Browse Categories in Regions
Srch_it_cat	0.004	0.028	Search Items in Categories
Srch_it_reg	0.0021	0.027	Search Items in Regions
view_items	0.004	0.0009	View Items
vu_usr_info	0.003	0.001	View User Info
vu_bid_hst	0.004	0.004	View Bid History

Table 4: CPU Service Demand for Each Component

nent in the middle (Business) tier and one in the Database Tier. Each component has its own resource requirements or Service Demands.

4.3 Customer Behavior Modeling Graph

For the initial experiment, the workload was composed of visitor type of clients. A typical user is expected to browse across the set of services and visit different sections of the auction site. A transition probability is assumed for a typical user to move from one service to the other.

The various transition probabilities are given in Table 5. Here element $p_{i,j}$ (at row i and column j) represents the probability of the i^th service being invoked after the j^th service is invoked. For example, a user in the web page “browse_cat”(browsing categories) has a 0.0025% chance of going to the “home” page and a 99% chance for moving on to "Search_it_cat"(searching for an item in a category).

	home	browse	browse_cat	browse_reg	br_cat_reg	Srch_it_cat	Srch_it_reg	view_items	vu_usr_info	vu_bid_hst	view_items_reg	vu_usr_info_reg	vu_bid_hst_reg	Probabilities
home	0	0.01	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0025	0.0026
browse	1	0	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0100
browse_cat	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0.0070
browse_reg	0	0.29	0	0	0	0	0	0	0	0	0	0	0	0.0029
br_cat_reg	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0.0029
Srch_it_cat	0	0	0.99	0	0	0.44	0	0.74	0	0	0	0	0	0.3343
Srch_it_reg	0	0	0	0	0.99	0	0.44	0	0	0	0.74	0	0	0.1371
view_items	0	0	0	0	0	0.55	0	0	0.8	0	0	0	0	0.2436
vu_usr_info	0	0	0	0	0	0	0	0.15	0	0.99	0	0	0	0.0747
vu_bid_hst	0	0	0	0	0	0	0	0.1	0.19	0	0	0	0	0.0386
view_items_reg	0	0	0	0	0	0	0.55	0	0	0	0	0.8	0	0.0999
vu_usr_info_reg	0	0	0	0	0	0	0	0	0	0	0.15	0	0.99	0.0306
vu_bid_hst_reg	0	0	0	0	0	0	0	0	0	0	0.1	0.19	0	0.0158

Table 5: Transition Probabilities Between Various Services

The steady state probability (percentage of user sessions) for each service type is denoted by the vector π. The value of π_i denotes the percentage of user requests that invoke the the i^{th} service. The vector π can be obtained by using a technique is similar to the one in [12]. Once computed, the amount of load on each service type can be calculated from the total number of user sessions. The rightmost column in Table 5 gives the steady state probabilities of each service.

4.4 Analytical Modeling of RUBiS Servlets

After the Service Demands and the steady state probability mix for each service is available, an analytical model of the application can be developed. The RUBiS benchmark assumes a client to carry out a session with multiple requests with think times in between. This type of a user behavior must be modeled with a closed model.

As soon as a client finishes, a new client takes its place. The average number of clients remains fixed. Figure 4 shows the analytical model of the RUBiS Servlets version. As mentioned in Section 4.2, the processor is the contentious resource. Each ma-

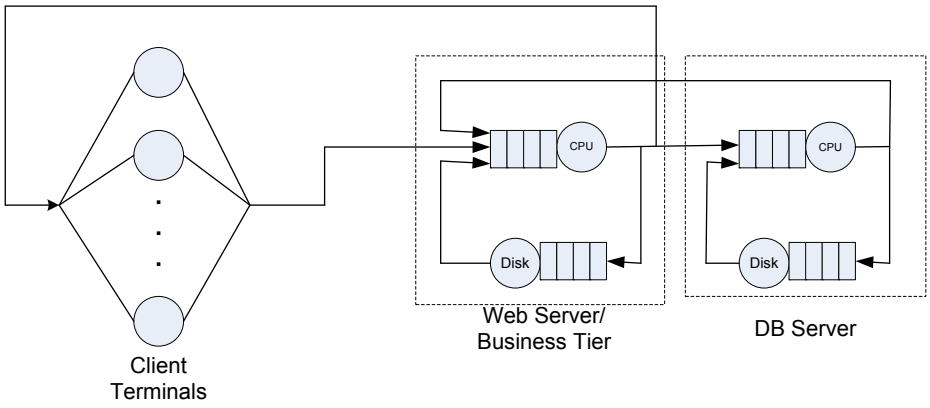


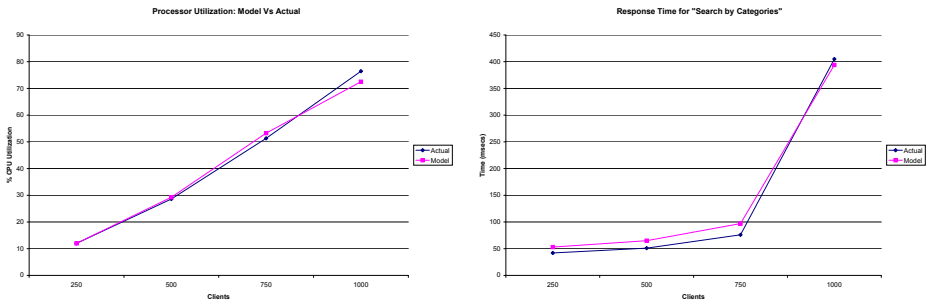
Fig. 4: Closed Queuing Model for Rubis Java Servlets Version

chine is represented by two queues, one for the CPU and the other for the disk.

Figure 4 also shows two queues for each of the two node in the deployment. The first node is the Business Tier, which also serves as the web server. The second node is the Database Server. The various client terminals are represented by delay servers. A delay server is a server that does not have a queue, so clients wanting to use the server can access it directly it without waiting. This design models user think times since as

soon as a response to a previous request comes back, the user starts working on the next request.

Figures 5a and 5b compare the results predicted by the analytical model to the actual results collected from running the benchmark. The benchmark is run using pro-



(a) CPU usage (b) Response Time

Fig. 5: **Validation of Analytical Model**

gressively increasing number of clients for $250, 500, 750$ and $1,000$, respectively. The components are placed in the nodes using RUBiS's default strategy, which places all the Business Tier components in the Business Tier node and the entire database in the database server. The results in these figures show the model accurately predicts the response times of the services and the processor utilizations of the nodes. This model can therefore be used by CAFe to find the placement of the components that optimizes the capacity of the deployment.

4.5 Application Component Placement

We now describe how CAFe iteratively places components onto hardware nodes. The SLA is assumed to have set an upper bound of 1 sec on the response time of all services. We use Algorithm 3 from Section 3, which considers CPU as the only resource since both memory and disk usage is minor compared to CPU usage, as described in Section 4.

The first iteration of Algorithm 3 uses the initial Service Demands for each application component. The Service Demands are given in Table 4. The set of all Service Demands and the available nodes (in this case 2) are used by the *Allocate* Algorithm 1. This algorithm in turn invokes the *worst_fit_bin_packing* described in Algorithm 2, which places components on the two nodes.

As mentioned in Section 4, there are two nodes used for RUBiS benchmark: Business Tier Server (BT_SRV) and the Database Server (DB_SRV). The name of the nodes are given since the default deployment of RUBiS uses the BT_SRV to deploy all the business layer components and DB_SRV to deploy the database. In fact, such a tiered deployment is an industry standard [13].

Table 6a shows the placement of the components after the first iteration of Algorithm 3 in CAFe. The mapping of the components to the nodes, the total number of clients (100), and the Service Demands of the components are used to build the analytical model. It is then used to find the response time and processor utilization of the two

BT_SRV		DB_SRV	
Component	CPU Util	Component	CPU Util
DB_SC	0.02783	DB_SR	0.02690
BT_VI	0.00405	BT_SC	0.00417
DB_BH	0.00400	BT_BH	0.00400
BT_UI	0.00300	BT_BCR	0.00325
BT_BC	0.00245	BT_BR	0.00253
BT_H	0.00200	BT_SR	0.00210
DB_UI	0.00100	BT_B	0.00200
DB_VI	0.00095	DB_BCR	0.00075
DB_BC	0.00055	DB_BR	0.00047

(a) Component Placement

Service	Business Tier Component%	DB Server Component%	Response Time
home	0.007	0.000	0.002
browse	0.029	0.000	0.002
browse_cat	0.025	0.005	0.004
browse_reg	0.010	0.002	0.004
br_cat_reg	0.014	0.003	0.005
Srch_it_cat	1.980	13.190	0.049
Srch_it_reg	0.380	5.260	0.041
view_items	1.940	0.490	0.006
vu_usr_info	0.480	0.120	0.005
vu_bid_hst	0.310	0.310	0.009

(b) Utilization and Response Time

Table 6: **Component Placement and RUBiS Performance After Iteration 1**

servers, given in Table 6b. The response time of all the services is well below the SLA specified 1 sec. CAFe iterates and the processor utilization of each component found in the previous iteration is used in the *Allocate* routine.

In the second iteration, the *Allocate* Algorithm 1 produces the placement shown in Table 7.

BT_SRV		DB_SRV			
Component	CPU Util	Component	CPU Util	Component	CPU Util
DB_SC	13.19	DB_SR	5.26	BT_B	0.029
		BT_SC	1.97	BT_BC	0.025
		BT_VI	1.94	BT_BCR	0.014
		DB_VI	0.49	BT_BR	0.010
		BT_UI	0.48	BT_H	0.007
		BT_SR	0.39	DB_BC	0.005
		BT_BH	0.31	DB_BCR	0.003
		DB_BH	0.31	DB_BR	0.002
		DB_UI	0.12		

Table 7: Iteration 2:Component Placement by Allocation Routine

In the third iteration, the number of clients, M is increase to 300. The placement computed by CAFe remains the same, however, and the response times of the two services "Search By Category" and "Search by Region" increase with each iteration as shown in Table 8.

Iteration	Clients	home	broBTe	broBTe_cat	broBTe_reg	br_cat_reg	Srch_it_cat	Srch_it_reg	view_items	vu_usr_info	vu_bid_hst
1	100	0.002	0.002	0.004	0.004	0.005	0.049	0.041	0.006	0.005	0.009
2	200	0.002	0.002	0.004	0.004	0.005	0.050	0.041	0.006	0.005	0.009
5	500	0.002	0.002	0.004	0.004	0.005	0.058	0.044	0.007	0.005	0.010
10	1000	0.002	0.002	0.005	0.005	0.006	0.088	0.049	0.007	0.006	0.011
15	1500	0.002	0.002	0.005	0.005	0.007	0.689	0.055	0.008	0.007	0.012
16	1600	0.003	0.003	0.006	0.006	0.007	1.119	0.057	0.008	0.007	0.012
17	1550	0.003	0.003	0.006	0.006	0.007	0.899	0.056	0.008	0.007	0.012
18	1575	0.003	0.003	0.006	0.006	0.007	1.011	0.057	0.008	0.007	0.012
19	1563	0.003	0.003	0.006	0.006	0.007	0.956	0.056	0.008	0.007	0.012

Table 8: Successive Iterations:Response Time of Each Service

At the value of $M = 1600$, the response time of the service "Search by Category" crosses the SLA limit of 1 sec as shown in iteration 16 in Table 8. At that point *incr*

variable in Algorithm 3 is reduced by half to 50 and M is reduced to the previous value of 1500. The algorithm continues from that point. Thus in iteration 17, value of M is

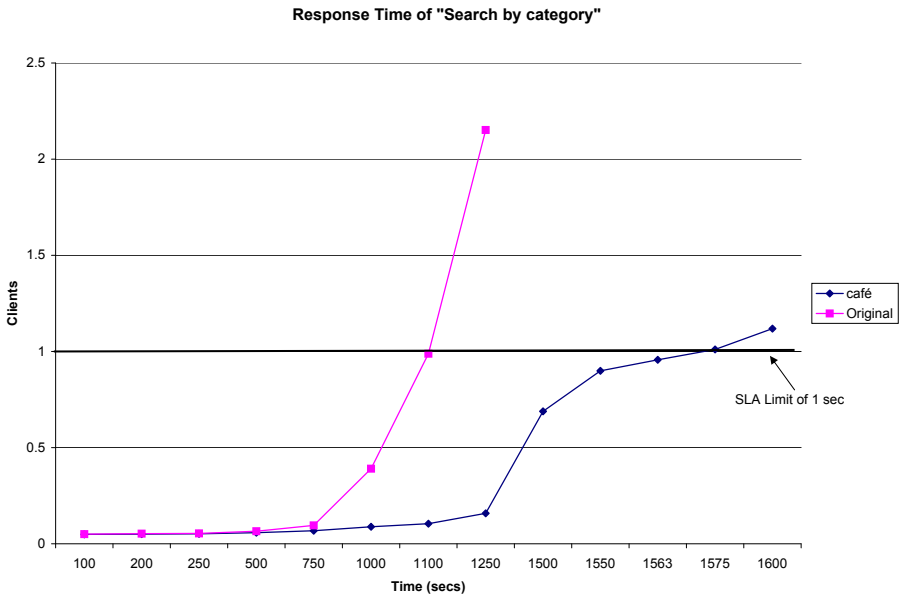


Fig. 6: Response Time with Increasing Clients

1550 In a similar way, for M equal to 1563, the response time of "Search by Category" is just below 1 sec (iteration 19). This response time is the maximum capacity of the application under a SLA response time of 1 sec. Figure 6 shows the comparison in the response time of the service "Search By Category," which is the bottleneck service.

4.6 Implementation of the CAFe Deployment Plan

We now describe how RUBiS uses the new deployment plan recommended by CAFe and empirically evaluate the performance improvement compared with the default tiered architecture used by RUBiS. This plan assigns all the Business Tier components in the BT_SRV and the entire database in the DB_SRV. The deployment suggested by CAFe is shown in Table 7, where component DB_SC is contained in one node and all the others are kept in the other node. The component DB_SC is the database component of the service "Search By Category," which is a read-only component that invokes a select query on the database.

One way to implement this assignment is to run a master instance of the database along with all the other components and run a slave instance of the database in the machine where DB_SC is run. The corresponding deployment is shown in Figure 7. In this figure there are two instances of the Database: the master instance is run in the machine BT_SRV and a slave instance is run in DB_SRV. All Business Tier components and the web server run in BT_SRV. These components make the database call on the master instance in BT_SRV. Only component DB_SC (which belongs to service "Search By

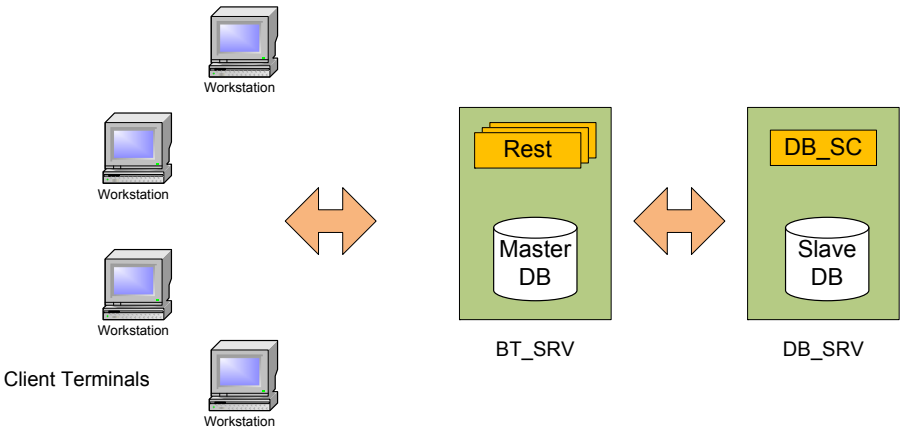
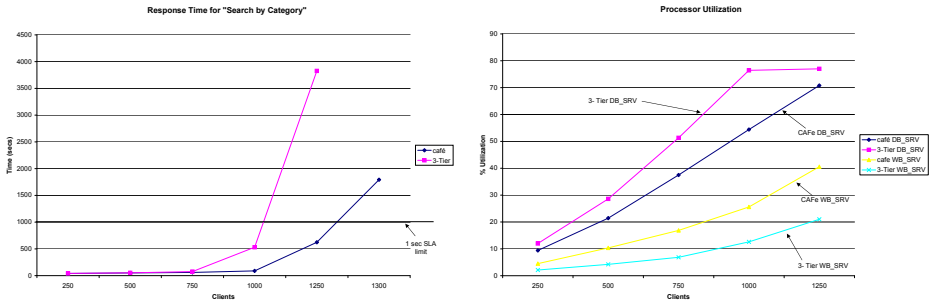


Fig. 7: Deployment of CAFe Suggested Assignment

Category") makes the database call to the slave instance (in DB_SRV). The component DB_SC is thus moved to DB_SRV, while all other components run in the BT_SRV.

Figure 8a shows the response times of the most loaded service "Search By Category" for the CAFe deployment. By comparison, the original response time with the



(a) Response Time (b) CPU Util

Fig. 8: **Performance of CAFe Installation**

3-Tier deployment is also provided. The comparison shows that the CAFe deployment increases the capacity of the application. The solid line (Time = 1.0) parallel to the Client axis signifies the SLA limit of 1 sec. The response times for the 3-Tier deployment crosses the line just above 1,000 clients. In contrast, the CAFe deployment the response time graph crosses the line at just over 1250 clients, which provides an improvement of $\sim$25% in application capacity.

Figure 8b shows the processor utilization for the two cases. In the CAFe installation the DB_SRV is less loaded than in the 3-Tier deployment. The WB_SRV utilization also shows the CAFe installation uses more CPU time than in the 3-Tier installation. This result is expected since CAFe tends to balance out the component utilizations across the given machines.

CAFe's balancing is not perfect,however, since DB_SC (the database component of the "Search By Category" service) consumes more processor time than all other com-

ponents. This result indicates that load balancing the DB_SRV on multiple components and moving the components to different machines may be advantageous.

5 Related Work

This section compares CAFe with related work in the area of system modeling and management.

Analytic models based on linear fitting. Stewart et. al. [1] proposed a profile-driven performance model for cluster based multi-component online services. They use this model to perform system management and implement component placement across nodes in the cluster. The main difference between CAFe and this work is that CAFe's modeling is based on queuing theory, whereas theirs is based on linear fitting. CAFe uses queuing theory since the impact of co-locating multiple components in the same node is better captured due to modeling of queuing delays. In [1]'s model, the impact of co-location is approximate.

Analytical modeling of multi-tiered applications have been pursued extensively in the research community. Closed queuing models of multi-tier internet application appear in [2, 3] and [12]. Both model a single tier as a queue and do not have any concept of a component in their model. CAFe leverages the knowledge of the components in the system and generates a queuing model out of the component placement mapping. Thus CAFe uses components as the functional granularity which has the advantage of utilizing the available resources in the nodes in a much better way, as shown in Section 2. The work in [3] presents techniques to predict workload that could be used along with CAFe.

A framework for dynamic placement of clustered web applications is presented by Karve et. al. [4] and Kimbrel et. al. [5]. These approaches consider multiple resources that are load-dependent and load independent. They solve an optimization problem that attempts to change the component placement at run-time, while minimizing the number of changes. They characterize resource requirements of components using a simple model that calculates Service Demands of different requests. Urgaonkar et. al. [6] identify resource needs of application capsule or components by profiling them and uses it to characterise application Quality of Service (QoS) requirements. They also define an algorithm for mapping the application capsules on to the platforms or nodes available. CAFe differs from both these work in terms of its workload and performance modeling. CAFe defines a queuing model which can model the interaction of several components co-located in a node. It also models the behavior of clients using Customer Behavior Modeling Graphs to characterize incoming workload. The analytical models developed in CAFe can be used along with the algorithms presented in [4, 5] or [6].

None of the work above (except [2]) has a concept of performance bound. CAFe introduces the performance bound through the concept of an SLA. The placement of the components is thus done to maximize capacity while ensuring that the performance remains within SLA bounds.

6 Concluding remarks

This paper presented a *Component Assignment Framework for multi-tiered Internet Applications* (CAFe), which is a novel algorithmic framework for mapping components of a multi-tiered application onto hardware nodes in web portals. CAFe helps ensure that (1) the capacity of the application is potentially maximized and (2) response times

remain within SLA prescribed bounds. CAFe complements research in the area of application placement by introducing the constraint of performance bounds. It also uses queuing-theoretic techniques to co-ordinate component placement and analytical modeling.

The paper also empirically evaluated CAFe against RUBiS, which is an industry-standard application benchmark. The experimental results showed how the CAFe Allocation algorithm can improve web portal performance by balancing the resource utilizations across various nodes. The performance improvement was 25%, which means that a representative web portal can handle more users without purchasing additional hardware. By using CAFe, therefore, earned revenue can potentially be increased by 25%.

References

1. Stewart, C., Shen, K.: 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) 71–84
2. Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: An analytical model for multi-tier internet services and its applications. SIGMETRICS Perform. Eval. Rev. **33**(1) (2005) 291–302
3. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic provisioning of multi-tier internet applications. In: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on. (2005) 217–228
4. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: Proceedings of the 15th international conference on World Wide Web, ACM New York, NY, USA (2006) 595–604
5. Kimbrel, T., Steinder, M., Sviridenko, M., Tantawi, A.: 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) 391
6. Urgaonkar, B., Shenoy, P., Roscoe, T.: Resource overbooking and application profiling in a shared Internet hosting platform. (2009)
7. Menascé, D.A., Almedia, V.A.F., Dowdy, L.W.: Performance by design: Computer Capacity Planning by Example. Prentice Hall, Upper Saddle River, NJ (2004)
8. Urgaonkar, B., Rosenberg, A., Shenoy, P., Zomaya, A.: Application Placement on a Cluster of Servers. International Journal of Foundations of Computer Science **18**(5) (2007) 1023–1041
9. Coffman Jr, E., Garey, M., Johnson, D.: Approximation algorithms for bin packing: a survey. (1996)
10. Schrijver, A.: Theory of linear and integer programming. Wiley (1986)
11. Amza, C., Ch, A., Cox, A., Elnikety, S., Gil, R., Rajamani, K., Zwaenepoel, W.: Specification and Implementation of Dynamic Web Site Benchmarks. In: 5th IEEE Workshop on Workload Characterization. (2002) 3–13
12. Zhang, Q., Cherkasova, L., Mathews, G., Greene, W., Smirni, E.: 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) 244–265
13. Eckerson, W., et al.: Three Tier Client/Server Architecture: Achieving Scalability, Performance and Efficiency in Client Server Applications. Open Information Systems **10**(1) (1995)