Shared Content Management in Replicated Web Systems: A Design Framework Using Problem Decomposition, Controlled Simulation, and Feedback Learning

Jingguo Wang, Raj Sharman, *Member, IEEE*, and Ram Ramesh, *Member, IEEE*

***Abstract*—Replication is one of the primary techniques used to improve the quality of distributed content service. It generally reduces user latencies and increases a site's availability. However, to our knowledge, there is no systematic framework that combines the structure of both content and service components of a Web application to design effective replica hosting architectures. Recent advances in interconnected and multiple content distribution network (CDN) architectures render this problem even more complex. In this study, we develop a systematic framework for designing and evaluating large-scale, component-based replication architectures for Web systems that are driven by both the quality and effectiveness of service provisioning on the service network. The proposed framework employs a combination of problem decomposition, configuration evaluation through controlled system simulations, and a neural-network-based feedback learning mechanism in the exploration of the design space. A case study demonstrates the viability of the framework. The framework can be an effective decision support tool for a system designer to systematically explore design options and select an appropriate design configuration that best meets the desired design objectives.**

***Index Terms*—Controlled simulation, decomposition, feedback learning, replica hosting system design.**

I. Introduction

THE DEMANDS on effective content distribution and delivery over the Internet have grown exponentially over the years. The landscape of content and other service offerings over the Internet is continuously evolving in response to growing and diverse needs for high-quality service at the user level and cost-effective service provisioning at the server level. In particular, the rapid increase in the size of content and the number of users has significantly impacted the way in which content is designed and distributed over the Internet [1]. The tremendous growth in demand for content is outpacing the capacity of the infrastructure of the Internet, as evidenced by the long response time encountered in many content delivering processes. According to Zona Research [2], if the response time to retrieve a Web page increases beyond 8 s, roughly 30% of the users abandon the retrieval process. Galletta *et al*. [3] show that increases in response time are clearly related to corresponding decreases in performance, attitudes, and the behavioral intentions of the users. Clearly, adding a new infrastructure throughout the Internet is not possible [1]. Considerable emphasis is placed on optimizing resource utilization in order to reduce the response time and improve the quality of service [4]. This is especially true in the case of commercial websites.

One of the primary techniques used to improve the quality of distributed content services is replication. By positioning content replicas optimally over the edge of the Internet, content delivery that might, otherwise, require traversal across the core of the Internet to the users could be simplified, expedited, and many of the latencies avoided [5]. Replication also leads to improved system availability through better fault-tolerance and better system scalability through the distributed provisioning of content and services. Such design configurations are known as *Replica Hosting Systems*, and the well-known content distribution networks (CDN) belong to this category [6], [7]. Popular CDNs include Akamai [8], Radar [9], and SPREAD [10].

Although the notion of content replication is intrinsically appealing from a performance perspective, the design and evaluation of such systems is still a great challenge. The recent advances in interconnected multiple-CDN architectures render this issue even more complicated [11]. The design space for such systems is large and seemingly complex [12]. To our knowledge, there is no systematic framework that combines the structure of content and service components of a Web system with the potential for replication in order to design effective replica hosting architectures.

A Web application can usually be described in three layers: *presentation layer, business logic layer*, and *database layer* (see, e.g., [13]). Each layer can be partitioned and distributed among the CDN's replica servers (or edge servers) [7]. Rabinovich and Spatscheck [5] describe a set of approaches for content (or service) replication at different levels. In such replication approaches, content elements drawn from the three layers are structured into *components* that are replicated. The components are then dynamically assembled and delivered from the replica servers when they are requested through technologies like Edge Side Includes [14]. Designing an effective replica hosting system is a multifaceted process due to considerations that must be accounted for such as the behavior of the users, the structure of the content, location of the edge servers, request routing,

Manuscript received July 11, 2006; revised March 2, 2007. This paper was recommended by Associate Editor J. Keane.

J. Wang is with the College of Business Administration, University of Texas, Arlington, TX 76013 USA (e-mail: jwang@uta.edu).

R. Sharman and R. Ramesh are with the School of Management, State University of New York, Buffalo, NY 14261 USA (e-mail: rsharman@buffalo.edu; rramesh@buffalo.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCC.2007.906049

network architecture, content consistency. Our objective in this research is to develop a systematic framework for designing and evaluating large-scale, component-based replication architectures for Web systems that are driven by the quality of service and effectiveness of service provisioning on the service network. More specifically, we address the following four design issues.

The first design issue pertains to the selection of content/service components for replication. An underlying assumption in this issue is that a Web system can be effectively partitioned into a set of independent but interacting system components. A simple example is a Web application that is a collection of *Site Views* where each view is composed of *Pages* and, in turn, each page consists of a set of *Data Components* drawn from an underlying database [15]. Such an architecture would yield a hierarchical organization of a Web system into components of increasing granularity in a bottom-up view. The second design issue is the placement of the component replicas among a set of distributed edge servers. When components are distributed, significant coordination among replica servers is needed in both content management and delivery. A compounding factor in the distribution strategy is the personalization needs of different user groups that could impose specific constraints on content distribution and management. The third design issue deals with the consistency maintenance policies for the component replicas. Coupled with the personalization needs, the consistency policies may directly impact the quality of service in terms of the data currency and the updating cost. The fourth issue pertains to the design of the network architecture for request routing and replica updating. By partitioning the underlying network structure into appropriate *network cells* and tracing replicas in the cells, we show that the request routing and consistency maintenance traffic could be decreased with a minimal tradeoff in the other performance metrics that are used in our case study.

The proposed framework employs a combination of problem decomposition, configuration evaluation through controlled system simulations, and a neural-network-based feedback learning mechanism in the exploration of the design space. The design parameters and performance metrics pertaining to these design issues are stochastic in nature and often lead to a combinatorial explosion in the design space. It is also compounded by difficulties in determining many of the metrics exactly. Exact optimization could quickly become intractable and, in the end, it may not be viable. Furthermore, the performance criteria are usually conflicting and closely connected. The tradeoffs among all metrics should be concurrently evaluated to achieve total system design, as an isolated subproblem-specific approach could easily become suboptimal. Hence, the completeness, robustness, and effectiveness of any exact optimization procedure for such large-scale system design problem could be difficult to establish. Heuristics are both preferred and widely used for the purposes of robustness and practicality [16].

The proposed design framework incorporates a *design through learning* strategy. The large-scale integrated design problem is first decomposed into specific manageable and interlinked subproblems. Next, the subproblems are solved through a combination of analytical and simulation modeling techniques. Exploring the design space with this approach provides results from controlled experiments to train a neural network. The trained neural network can, then, be used to predict potentially effective design configurations for further exploration, if necessary. The neural network mechanism also demonstrates the tradeoffs among the performance metrics according to different design parametric settings. Consequently, the framework presents an effective decision support tool for a system designer in that it systematically explores design options and selects an appropriate design configuration that best meets the design objectives. The framework incorporates an integration of several techniques such as cluster analysis, perturbation analysis, Tabu search, simulations, and neural-network-based feedback learning. This approach follows a recent trend toward using simulation modeling for large-scale system optimization by integrating with several related analytical methods [17]–[22].

The organization of this paper is as follows. Section II reviews related work. Section III presents the design environment and includes an explanation of the performance metrics that have been developed and a description of the exogenous and endogenous parameters. Section IV details the solution framework that includes the problem decomposition strategy and the underlying algorithms. The experimental results are discussed in Section V and the concluding remarks are presented in Section VI.

II. Related Work

A replica hosting system is an effective approach to improve the quality of Internet service. The system replicates the content from the place of origin to the replica servers strategically positioned over the edge of the Internet and serves a request from a replica server close to where the request originates. An overview of CDN architecture is given in [23]. Sivasubramanian *et al*. [12] provide a general framework for replica hosting systems and a comprehensive survey of different Web hosting systems. The survey compares the different strategies adopted by four different CDNs and includes their merits and demerits.

In general, five issues need to be considered in the design of a Web hosting system: metrics determination, adaptation triggering, replica placement, request routing, and consistency enforcement. We summarize recent work in each of these domains as follows.

1) *Metric Determination*: The question addressed in this issue is how to find and estimate the system performance metrics. In general, there are five classes of metrics used to evaluate performance in replication hosting systems, including temporal metrics [24], [25], spatial metrics [26], [27], network usage metrics [12], financial metrics [28], and consistency metrics [12].
2) *Adaptation Triggering*: The question addressed in this issue is when to adapt the system configuration to improve the system performance. Adaptation triggering mechanisms can be classified based on their temporal nature and these classifications include periodic triggers, aperiodic triggers, and hybrid triggers [9]. The mechanisms can also be classified according to where the adaptation is initiated, and these classifications include server-triggered

adaptation [9], client-triggered adaptation [29], and router triggered adaptation [10].

3) *Replica Placement*: The question addressed in this issue is where to place replicas. Two widely studied subproblems are replica server placement [30], [31] and replica content placement [9], [10], [32], [33].
4) *Request Routing*: The question addressed in this issue is how to direct users to the replica they need. A request routing system can be nonadaptive or adaptive [9], [10], [34], [35]. Further, it can be implemented transparently or nontransparently [9], [10], [34]–[36].
5) *Consistency Enforcement*: The question addressed in this issue is how to keep the replicas of given objects consistent. Three types of consistency mechanisms are studied, including: time-based consistency [37], value-based consistency [38], and order-based consistency [39]. The direction of the update can be pull-based [37], push-based [40], or hybrid [8], [38].

Although there is a vast body of literature on replica hosting systems [1], [6], [8], [23], [28], [30], many of these studies focus on a single issue and assume other issues to be independent. To our knowledge, there is no systematic framework that combines the structure of content and service components of a Web system with the potential for replication to design effective replica hosting architectures. According to Sivasubramanian *et al*. [12], the design and evaluation of content replication systems is still a great challenge from a performance perspective, and the design space for replica hosting systems is large and seemingly complex. Our focus in this research is to develop such a framework, one that is driven by the quality of service and effectiveness of service provisioning on the service network.

III. Design Environment

Designing a replica hosting system involves several complex, interrelated optimization issues. The underlying optimization process can be described using the feedback control configuration described in [12]. The system under consideration can be modeled using an abstract objective vector Y, which is a function F of a set of exogenous and endogenous parameters $p_1, p_2, \ldots, p_n$. Exogenous parameters are the environmental factors that define the scope and scale of a system, whereas endogenous parameters are specific controllable design variables.

The nature and form of function F that yields the objective vector Y are complex and difficult to determine explicitly. Consequently, an iterative scheme of evaluation for F is needed. In the scheme, F is first evaluated from a set of initial parameters. Subsequently, the endogenous parameters are adjusted in the feedback loop using the resulting Y, and this process is repeated until Y converges to the desired levels of performance. The values of endogenous design parameters that will result in the desired level of performance are, thus, determined. In the following discussion, we develop the system objectives [structure of Y and the underlying parameter sets (see Fig. 1)].

A. System Objectives

Our design objectives are to: 1) maximize the quality of service (QoS) delivered to users; 2) minimize operational and maintenance costs; and 3) maximize the currency of the content delivered to the users. These design objectives and their associated metrics are described in Table I.

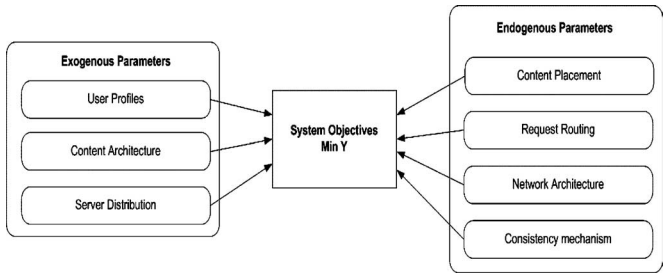


Fig. 1. Problem structure.

TABLE I
Performance Metrics

Notations	Description
y_1	Average response time to user requests (Metric for the QoS)
y_2	Content updating cost (Metric for operational and maintenance costs)
y_3	The Number of Multicasts sent per period to locate content (i.e. Request Routing). (Metric for operational and maintenance costs)
y_4	Traffic intensity between replica servers (Metric for operational and maintenance costs)
y_5	Staleness Index, measuring the currency of the content (ratio of obsolete content delivered to the total number of requests)

The metric y_1 measures the average latency time of user requests. The latency includes network communication delay and server service delay [24], [25]. The metric y_2 measures the overhead introduced to keep the content consistency and is determined as follows. Suppose that a unit of content is updated w times per s and that the updates are always immediately pushed to k replicas. If the average path length for a connection from the server to a replica is l, then the update cost is considered to be $w \times k \times l$ [12]. The metric y_3 measures the traffic generated by request routing. It is determined by the number of multicast messages sent in a given period for locating content. The traffic intensity metric y_4 is another measure of network usage. The traffic from location i to location j is measured by the size of the content required by location j from location i over a given period of time. The staleness index y_5 measures the currency of content delivered as the ratio of obsolete content delivered to the total number of requests. Each metric is a complex, stochastic function of the design parameters. Our design objective is to minimize Y, where $Y = (y_1, y_2, y_3, y_4, y_5)$, subject to a set of design constraints.

B. Exogenous Parameters

The exogenous parameters considered in this study include user profiles, content architecture, and server distribution over the Internet. Table II summarizes the notations for exogenous parameters.

1) User Profile: The user profile parameters add personalization needs in content delivery. Such customization of content is based on the user's location and access behavior. Central to

TABLE II
EXOGENOUS PARAMETERS

Notation	Description
User Profiles	
CB_l	The number of users from location l, $l \in L$, $CB = \sum_{l \in L} CB_l$
SCM	Segment composition matrix, *SCM(s,l)* denotes the probability of segment (user profile) s from location l.
Content Architecture	
C	Component set, $c \in C$
P	A page $p \in P$ is such that $p \subseteq C$
S	A user profile or a site view $s \in S$ is such that $s \subseteq P$
f_c	Expected update frequency of component c
r_c	Access pattern of component c, $r_c = (r_{c,1}, r_{c,2}, ..., r_{c,\|L\|})$. $r_{c,l}$ is the number of read accesses made from location l for component c, $l \in L$
G	Component group set, and a component group $g \in G$
Server Distribution	
L	Location set, $l \in L$
$D(l',l)$	Distance between location l' and l, $l', l \in L$
$TR(l',l)$	The traffic from location l' to l, $l', l \in L$

TABLE III
SEGMENTATION COMPOSITION MATRIX

Locations	User Profiles	
	s_1	s_2
1	50%	50%
2	30%	70%
3	60%	40%

personalization is the notion of segmenting users into groups according to their profiles. User segmentation has been well studied in the marketing literature [41]. In particular, each user group is associated with a well-specified profile that determines their content needs, access privileges, etc. The user segments and their associated profiles are specified as follows in our context. Let L denote the set of locations of replica servers. Let CB denote the total customer base of a replicated application system measured by the number of users. The customer base in each location is denoted as CB_l, $l \in L$. We have $\mathrm{CB} = \sum_{l \in L} \mathrm{CB}_l$. Let S denote the set of user segments, and $s \in S$ denote an individual segment. Each segment is associated with a unique profile. Note that each location could serve multiple segments and each segment could occur at multiple locations. The probability that segment s occurs in location l is denoted as *SCM(s, l)*.

Example. Consider a Web application with two user profiles (s_1 and s_2). Further assume that three locations (1–3) need to access the application with a rate of 450 requests per hour and that the access for locations 1, 2, and 3 is 22%, 33%, and 44%, respectively. Table III shows the distribution of user profiles by location.

2) Content Architecture: Content architecture is central to replication strategies in Web applications. In our study, we employ a component-based model for content architecture, one that

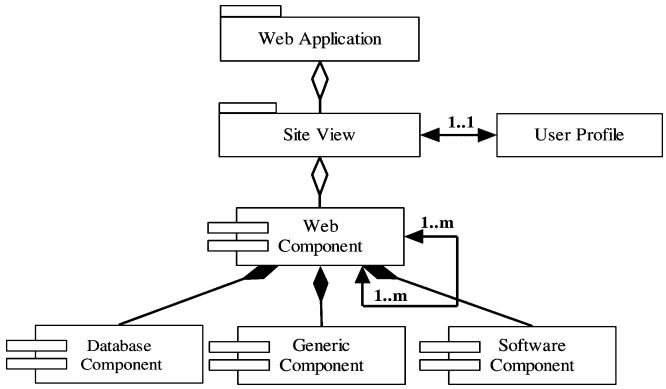


Fig. 2. Component-based content architecture.

aids in decisions about content selection and placement. The purpose of the proposed hierarchical model of content architecture is to describe the structure of the content and to characterize the user behavior using content components. Component-based hypertext context architectures such as WebML [15] have already been developed. A component model for entities that are comprised of markup language (HTML, XML) elements is introduced in [42]. Content management systems such as CoreMedia AG (http://www.coremedia.com/) adopt object-oriented content architecture for management and delivery. Dutta *et al*. [43] propose an optimization model for the design-time decision to determine which objects should be candidates for caching based on the hierarchical relationships among the objects for object-oriented applications.

A high-level view of the content architecture is presented in Fig. 2. A Web application system is modeled as a collection of site views where each site view addresses a specific user segment. While the mapping between site views and user profiles could be many-to-many, we employ a one-to-one mapping for the sake of brevity in our exposition. Each site view is a collection of interlinked Web pages where the links could be either navigational or more system-specific architectural connections. A page is a special type of content component. Each page presents a set of nested components. Typically, components are classified as either static or dynamic [5]. While the static components are preassembled and ready for delivery, the dynamic components are generated "on the fly" according to user requests. In commonly known content architectures, the static components are delivered to every user of a site view whereas the dynamic components are instance-specific, as required by a user's query. Finally, a content component could be a collection of instances of database entities (as needed in queries), or a complete database entity itself, or any generic content that does not belong to any underlying database schema (such as a banner item, for instance). Besides the regular content, components could also be software elements such as applications that run on top of a database, interface components (such as Java Server Page, Components, or Java-script code, for instance), or communication control elements (such as agents and protocols). For the sake of simplicity, we refer to all these elements as "components" in our formal specification given later.

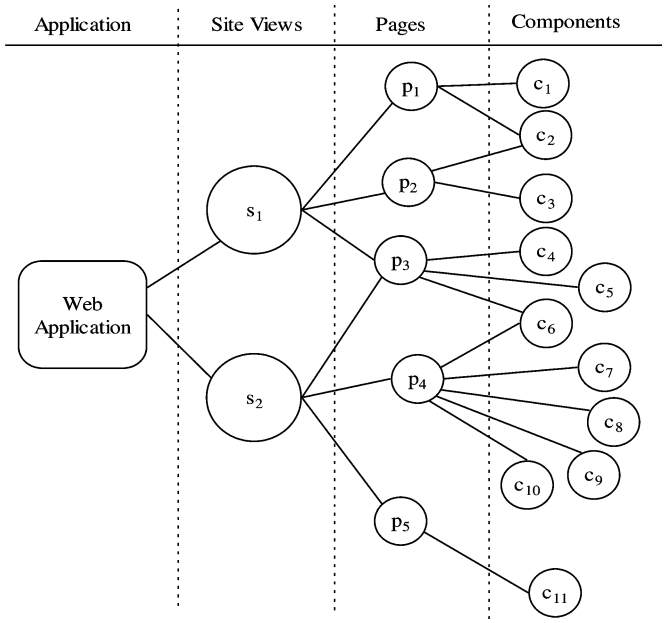


Fig. 3. Content architecture.

TABLE IV
COMPONENT ACCESS PATTERN (THE VOLATILITY OF THE FREQUECNY PER HOUR)

		Locations		
		1	2	3
Components	c1	50	45	120
	c2	100	90	240
	c3	50	45	120
	c4	100	150	200
	c5	100	150	200
	c6	150	255	280
	c7	50	105	80
	c8	50	105	80
	c9	50	105	80
	c10	50	105	80
	c11	50	105	80

TABLE V
DISTANCE BETWEEN LOCATIONS (IN THOUSAND MILES)

		Locations			
		1	2	3	O
Locations	1	0	15	20	25
	2	15	0	10	35
	3	20	10	0	30
	O	25	35	30	0

TABLE VI
ENDOGENOUS PARAMETERS

Notation	Description	Categories
L(c)	Replica locations of component c, $c \in C$, $L(c) \subseteq L$	Content Placement
k	Number of Component Groups	Content Placement
n	Number of network cells	Request Routing; Network Architecture
t	Strong Consistency Threshold	Consistency Mechanisms
t	Time Interval for Updating	Consistency Mechanisms
b	Content Maintenance Budget	Budget Constraint

We define the architecture using a bottom-up scheme as follows. Let C denote the set of all the components and $c \in C$ represent an individual component. We assume that each component is uniquely self-contained, has necessary interfaces for access and activation, and can be replicated. Let P denote a set of pages and $p \in P$ including the set of components composing a page. Similarly, let S denote the set of site views and $s \in S$ including the set of pages composing a site view. Each site view corresponds to one user segment. Note that the mapping between site views and pages is many-to-many. Similarly, the mapping between pages and components is also many-to-many. The volatility of the frequency of updates to component c is denoted by f_c. For ease of presentation, we assume each component to be of unit size in the following analysis.

Example (Continued). Fig. 3 shows the content architecture of the application. The architecture contains two site views (s_1, s_2) corresponding to two user profiles. $s_1 = \{p_1, p_2, p_3\}$, and $s_2 = \{p_3,\ p_4, p_5\}$. p_i is a page. Further $p_1 = \{c_1, c_2\}, p_2 = \{c_2, c_3\}, p_3 = \{c_4, c_5, c_6\}, p_4 = \{c_6, c_7, c_8, c_9, c_{10}\}$, and $p_5 = \{c_{11}\}$. c_i is a component. The update frequency per h (f_c) for each of the components $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}$ is assumed to be 10, 7, 12, 6, 3, 2, 4, 5, 2, 3, and 3, respectively. Based on the segment composition matrix (SCM) (Table III) and content architecture (Fig. 3), the access pattern of components is calculated in Table IV.

3) Server Distribution: Users in each location are served by a local replica server. The service areas of replica servers are nonoverlapping, and the replica servers access each other either through peering or public networks. The origin server is located in the Internet, and the topographical distances between replica servers are assumed to be much smaller than those between the origin and replica servers. $D(l', l)$ denotes the distance between two servers l', $l \in L$. Table V shows the distance between locations for the example. There is a local replica server in each location.

C. Endogenous Parameters

The endogenous parameters that we have considered in this study for a replica hosting system are: 1) content placement, representing content selection and placement among replica servers; 2) request routing, specifying the routing mechanisms to direct user requests to the appropriate replica servers; 3) network architecture, specifying the grouping of replica servers into network cells, the internal architecture of the cells, and coordination mechanisms; 4) content consistency mechanism, dealing with how to keep the replicas of objects consistent; and 5) content maintenance budget. A summary of the endogenous parameters is presented in Table VI.

Each network cell consists of a metaserver that coordinates content requests from users and delivery from replica servers in the cell. The metaservers of different cells communicate among

each other in locating content. Grouping replica servers into cells and using metaservers to coordinate content requests are to reduce the traffic between the replica servers for content locating and to improve the overall performance of the system.

The communication between metaservers in replica server cells is through multicast for request routing, and the following protocol is used.

1) The user's page request is first directed to the local replica server and it is referred to as the initiating server.
2) The initiating server parses the page request into a component-requesting list.
3) For the components that are not locally replicated, the initiating server sends a query to the metaserver in its cell.
4) The metaserver checks whether the requested component is replicated in the cell. If the component is not replicated in the cell, the metaserver multicasts the request to other metaservers as well as to the origin server.
5) Metaserver(s) forward the request to the replica servers having the requested component.
6) The servers that have the requested component reply to the initiating server.
7) The initiating server selects the server that responds first.
8) Through a protocol exchange, the requested components are sent to the initiating server.
9) The initiating server assembles the components into the page and returns to the user.

The origin is the primary server for all content and maintains the newest version. Content updates are initiated by the origin server and, through metaservers, the new content is pushed to corresponding replica servers. Maintaining consistency among replicas adds a significant overhead, especially when strong consistency mechanisms are enforced and the number of replicas is large. A hybrid, time-based consistency model [44] is hypothesized in our study. We define a threshold to classify components into two categories based on their read-to-write ratio [45]. Strong consistency is maintained for the components that have a read-to-write ratio above the threshold τ, and updates on these components are always immediately pushed to replica servers. A time-interval-based consistency is employed for other components, and updates are pushed periodically with an interval t.

IV. Design Framework

We decompose the problem of large-scale design optimization into four interdependent subproblems as shown in Fig. 4. Each subproblem involves a specific set of design objectives and its own solution space and solution method. The design objectives for each subproblem constitute either a subset of the overall objectives or certain surrogates. The interlinking of the solution follows an input–output structure, where the solution of a subproblem constitutes part of the inputs to the succeeding subproblem solved in the sequential iterative framework. The subproblems are linked in an iterative loop in the overall solution framework and this loop, in turn, incorporates feedback and learning mechanisms. A neural network is used to learn the characteristics of the design space from the solutions that are generated. The progressively trained neural network can be successively employed in further exploration of a design space. We develop the subproblem structures and associated algorithms in the following discussion.

A. Component Grouping

Since the number of components is usually very large, componentwise replica placement and tracking during access can be computationally prohibitive. A clustering of components into groups, followed by their collective groupwise replication, would lead to a more effective placement and tracking strategy. This approach would also entail tradeoffs between content management costs and user access performance. Improved access performance at relatively low management costs with such a grouping strategy has been shown in the literature [32]. Furthermore, Chen *et al*. [32] propose several clustering schemas, and, among these, they propose spatial clustering which clusters content based on the spatial locality in the access patterns and has the best performance.

In our proposed framework, the components are clustered with similar access patterns. The read frequency pattern of component c is denoted as $r_c = (r_{c,1}, r_{c,2}, \ldots, r_{c,|L|})$, where $r_{c,l}$ is the number of read accesses made from location l for component $c, l \in L, c \in C$. The calculation of $r_{c,l}$ is based on the Bayesian rule with the information from customer base composition, user profile, and component composition hierarchy. The components are grouped by a k-means clustering algorithm [46]. Let G denote the set of component groups, and g denote a specific group with $g \in G$. The distance between two components c and c' is measured by the Euclidean distance between the vectors r_c and $r_{c'}$ in the $|L|$-dimension space, denoted as $\mathrm{ED}(c, c')$. The distance between a component group g and component c is defined as the average Euclidean distance between c and the components in g : $E(c, g) = \frac{\sum_{c' \in g} \mathrm{ED}(c, c')}{|g|}$. A component will be clustered into a group whose distance is the smallest. The determined component groups constitute a set of inputs to the next subproblem, component placement. Let k denote the number of component groups. Choosing an integer range $\mathrm{LB} \leq k \leq \mathrm{UB}$, where LB is the lower boundary and UB the upper boundary of k, the algorithm may be performed for each value of k in this range. The optimal k and its associated clusters are determined based on the performance metrics Y.

Example (Continued). We consider clustering components into groups. The number of component groups k needs to be determined. There may be a range of possible values for k. Let us consider $k = 4$. The k-means clustering algorithm yields g_1, g_2, g_3, and g_4 given by $\{c_7, c_8, c_{10}, c_{11}\}$, $\{c_2, c_4, c_5\}$, $\{c_1, c_3\}$, and $\{c_6\}$, respectively.

B. Component Placement

The objective of component placement is to place the component groups in an ideal location so that the average response time can be minimized within an updating cost budget constraint b. A sensitivity analysis on b would provide the insight of the tradeoff between update costs and query costs.

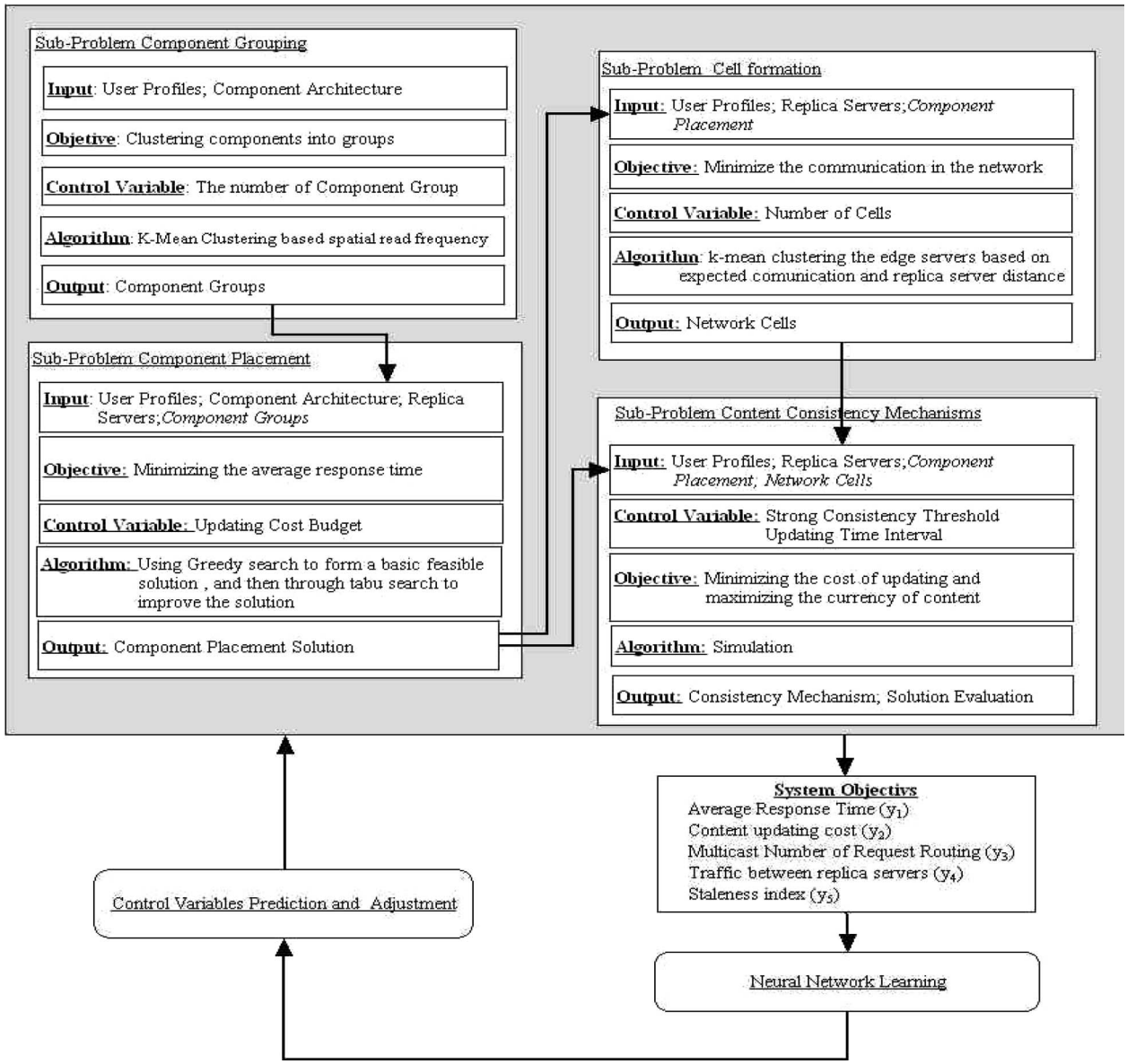


Fig. 4. Decomposition strategy.

While placement solutions can be evaluated through simulation, they are computationally expensive. Consequently, metamodel surrogates of the placement objectives are used to estimate the quality of placement solutions. The extent to which geographical distance can be used as a surrogate for latency metrics is studied in [26], and a fairly close correlation between geographical distance and round-trip time is shown. Consequently, we use the distance between two servers as a proxy measure of content transmission time between them. The metamodel to determine the estimated average response time $\hat{y}_1$ is shown in Table VII. The service time of a server is assumed to be relatively small compared to the transmission time in this metamodel. A strong consistency mechanism is assumed in the determining of the estimated updating cost $\hat{y}_2$. Now, suppose the expected updating frequency of a component c is f_c per unit time and $L(c)$ is the location set where component c is replicated. The update costs are, then, estimated as $\sum_{l \in L(c)} f_c \cdot D(l, o)$, where $D(l, o)$ is the distance between replica server l and the origin server and represents the content transmission cost. With the metamodels, we obtain a potentially effective solution for further analysis.

A greedy content placement algorithm is proposed in [32]. A modified version of this algorithm used in the paper is presented in Table VIII. Note that the objective function is implicit and complex. Using the solution obtained by the greedy algorithm as a starting point, we conduct a Tabu search to improve the solution.

The Tabu search is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality [47]. The strategy of the Tabu search is to avoid being trapped into cycles by forbidding moves that iterate back to already explored solution spaces. To avoid retracing steps, the method

TABLE VII
METAMODEL OF ESTIMATED AVERAGE RESPONSE TIME $\hat{y}_1$ FOR COMPONENT PLACEMENT

// Meta Model Of Estimated Average Response Time $\hat{y}_1$.

input: $\{L(c) \mid c \in C,\ L(c) \subseteq L\}$: a content placement solution.

output: Estimated average response time $\hat{y}_1$.

begin
  // for each component
  for $\forall\ c \in C$
    // for each location
    for $\forall\ l \in L$
      //*ARTC(c,l)*: average response time for component *c* in location
      // *l*. The nearest replica server that has the requested component
      // is always selected. The transmission time is approximated by
      // the server geographical distance.

$$ARTC(c,l) = \min_{l' \in L(c)} D(l', l)\ ;$$

    end;
  end;
  //*ART(l)*: average response time in location *l*, which is weighted
  // average of *ARTC(c,l)* with the access frequency of the components
  // as weights.

$$ART(l) = \frac{\sum_{c \in C} ARTC(c,l) * r_{c,l}}{\sum_{c \in C} r_{c,l}}\ ;$$

  //*ART*: average response time, which is an average of *ART(l)*.

$$\hat{y}_1 = \frac{\sum_{l \in L} ART(l)}{|L|}\ ;$$

end;

TABLE VIII
GREEDY ALGORITHM FOR BASIC FEASIBLE SOLUTION FOR COMPONENT PLACEMENT

// Basic Feasible Solution
// A greedy algorithm
input: b: Updating Budget Cost
output: $\{L(c) \mid c \in C,\ L(c) \subseteq L\}$: a content placement solution
begin
  //for all groups
  for $\forall\ g \in G$
    //initially all groups reside at the origin server
    for $\forall\ c \in g \quad L(c) = \{o\}$ end;
  end;

  // the estimated updating cost $\hat{b}$ of the current solution is 0;
  $\hat{b} = 0$;
  while $\hat{b} < b$;
    for $g \in G$;
      for $l \in L$;
        calculate the estimated average response time $\hat{y}_1$ (table
        4) and the estimate updating cost $\hat{y}_2$ using
        $\sum_{l \in L(c)} f_c \cdot D(l,o)$ if group g is replicated to l;
      end
    end

    choose a pair of group and location <*g, l'*> with smallest $\hat{y}_1$
      and $\hat{y}_2 < b$;
    if we can't choose such as solution;
    // Components can't be replicated anymore given budget constraint
      exit while;
    end
    for $c \in g \quad L(c) = L(c) \cup \{l'\}$ end;
    set $\hat{b}$ to the updating cost $\hat{y}_2$ of the chosen solution;
  end
end

records recent moves in Tabu lists (Tabu memory) that force the search to explore new areas of the solution space. Furthermore, at the point of initialization, the solution space is evaluated using a process of "diversification," which is used to direct the search from regions previously explored by degrees that are strategically different. As candidate locations and local optima are found, the search becomes more focused in a process known as "intensification." Intensification may return to previously explored "good regions" to look more thoroughly or seek to incorporate compatible attributes of previous good solutions into current solutions.

We use the solution obtained by the greedy search as a basic feasible initial solution for the Tabu search. The Tabu search algorithm consists of the following elements: *Elite Candidate List, Moves, Tabu List*, and *Aspiration Criteria* [47]. An *elite candidate list* for iterations is formed by selecting the best K (elite) solutions within each iteration that are evaluated in one subsequent iteration. Typically, in each evaluation, K candidate solutions are generated from a given initial solution. Next, we select K best solutions among the $K \times K$ candidates and proceed with these in the next iteration. This procedure results in the evaluation of exact K solutions in each iteration. We employ three *moves* as follows: exchange of two component groups between two locations, shift of one component group from one location to another, and addition of a component group to a location where it is not replicated. The solutions are evaluated with the estimated updating cost and the estimated average response time. When a move is not feasible, i.e., the estimated updating cost of the solution generated by the move is greater than the updating cost budget, $\hat{y}_2 > b$, a penalty is added to the estimated average response time $\hat{y}_1$ of the solution. The *Tabu list* is used to prevent a sequence of moves that will cycle among a set of solutions. These are implemented by disallowing repeat exchanges in T steps whenever an exchange/shift/add has been applied to component groups and locations. We define the *Aspiration criteria* as the condition to accept a Tabu active move, or as the move which produces a solution that has a better estimated average response time than the current best solution. Finally, restarting can effectively help prevent searches from repeating. If any iteration fails to produce K acceptable candidates selected from the current iteration, then we restart. Restarting can be done with the K elite solutions either from the previous iteration or from preceding iterations.

Example (Continued). We now place the component groups in three edge servers given $b = 40$ unit cost per hour. The greedy search algorithm is used initially and yields an estimated average response time of 6.73 s and an estimated updating cost of 35.00 unit cost per hour. The greedy algorithm results in the

TABLE IX
TRAFFIC PATTERN BETWEEN EDGE SERVERS

		Locations		
		1	2	3
Locations	1	12.5	31.25	56.25
	2	10.71	62.50	76.79
	3	30.77	51.28	117.95

following content placement solution: groups $\{g_4\}$, $\{g_1, g_2, g_4\}$, and $\{g_3, g_4\}$ are placed in locations 1, 2, and 3, respectively. Refinement with Tabu search results in the placement groups $\{g_2, g_3, g_4\}$, $\{g_1, g_4\}$, and $\{g_2, g_4\}$ at locations 1, 2, and 3, respectively. The estimated average response time and the estimated updating cost improves to 4.83 and 38.68, respectively.

C. Cell Formation

Given user profiles, server distribution, and component placement solutions, we form network cells based on the distances and expected traffic between replica servers. A k-means clustering algorithm is used. The solution to the cell formation problem is a grouping of replica servers into an optimal set of cells, which is an input to the next subproblem: content consistency mechanism design. Given a range, the optimal number of network cells (n) and its associated formation are determined based on the performance metrics Y.

The expected traffic is determined from the user profiles and the placement of components derived from the Tabu search procedure. In the traffic estimation, we assume that the content will be brought from the nearest available location. The traffic from location l to l' is approximated by the amount of content that will be sent from location l to l'.

We employ the following estimate for intermetaserver distances. Note that one metaserver is placed in each cell. The distance between the metaserver of a cell and the replica servers in the cell is the average distance between the replica servers in the cell. The distance of a metaserver to the origin is the average distance of the replica servers in the cell to the origin. The distance between any two metaservers is the average distance between the replica servers in these two cells.

Example (Continued). With the content placement solution, we calculated the expected traffic between edge servers as recorded in Table IX. We group the replica servers into two cells with k-means clustering based on expected communication (Table IX) and geographical distance (cell 1, location 2; cell 2, locations 1 and 3).

D. Content Consistency Mechanisms

An important issue in distributed systems concerns the need to ensure that the replicated components are the same. This issue of content consistency has been extensively researched in the area of distributed computing and there are several consistency models that have been developed [45]. In this study, we use a hybrid, time-based consistency mechanism. We first classify components based on the following rule:

1) Component Class 1: $C1 = \{c | \sum_l r_{c,l}/f_c \geq \tau,\ c \in C\}$, which includes all components having a read-to-write ratio above or equal to the threshold τ;
2) Component Class 2: $C2 = \{c | \sum_l r_{c,l}/f_c < \tau,\ c \in C\}$, which includes all components having a read-to-write ratio below the threshold τ.

The classification allows us to choose different update strategies for different component classes. For the components in class $C1$, their updates will be immediately pushed to the replica servers. For the components in class $C2$, their updates will be periodically pushed to the replica servers in the time-interval t.

In this subproblem, we determine the threshold τ for component classification and the update interval t for Class $C2$. These two parameters are evaluated through a controlled simulation. The controlled simulation entails simulation experiments with system configuration using various levels of τ and t. Note that in order to perform simulations, a system configuration consisting of component groups, component group placement, and the cell structure of the replica servers is needed. We may obtain different configurations by solving the previous three subproblems with different levels of the number of component groups (k) and the number of network cells (n). The overall performance metrics $Y = (y_1, y_2, y_3, y_4, y_5)$ are evaluated in the simulation experiments. The detailed design of the controlled simulation experiments is described in Section V.

E. Feedback and Learning Mechanisms and the Integrated Solution Framework

The final stage of the proposed design framework incorporates a supervised learning mechanism by using a neural network. The neural network is trained to learn about the effect of the endogenous and exogenous parameters on the overall system performance and, consequently, it can be used to predict system behavior under a wide range of system configurations.

We use a two-layer feedforward fully connected neural network. The inputs of the network are the values of the control variables, including the number of component groups (k), the updating cost budget (b), the number of network cells (n), the strong consistency threshold (τ), and the time interval for updating (t). The outputs of the neural network are the performance metrics Y obtained through simulations. We did not include the traffic between replica servers as output of the neural network since it can be approximately measured by the number of multicasts for locating content in the network. Cross-validation techniques [48] are used to decide on the number of hidden layers and their sizes. The transfer function in the hidden layers is tan-sigmoid, and the output layer transfer function is linear. The Levenberg–Marquardt algorithm [49] is employed for network training. In general, with such problems, it has been shown that the algorithm provides the fastest convergence [50].

In our integrated design framework, problem deconstruction and the sequentially integrated solution of the subproblems are essential to identify potentially effective design configurations. By running a subset of possible configurations of control variables, enough observations are obtained to train the neural network to predict system behavior. Repeating these trials over

different ranges of parametric settings, the trained neural network can be used to yield predictive support for a wide range of configurations in practical design.

V. Results and Discussion

This section is devoted to a detailed discussion of the results in terms of the different performance metrics used in this study. At the outset, in Section V-A, we discuss the experimental design along with different levels of parameter settings. The section ends with a discussion relating to the feedback and learning systems using neural networks.

A. Design Environment

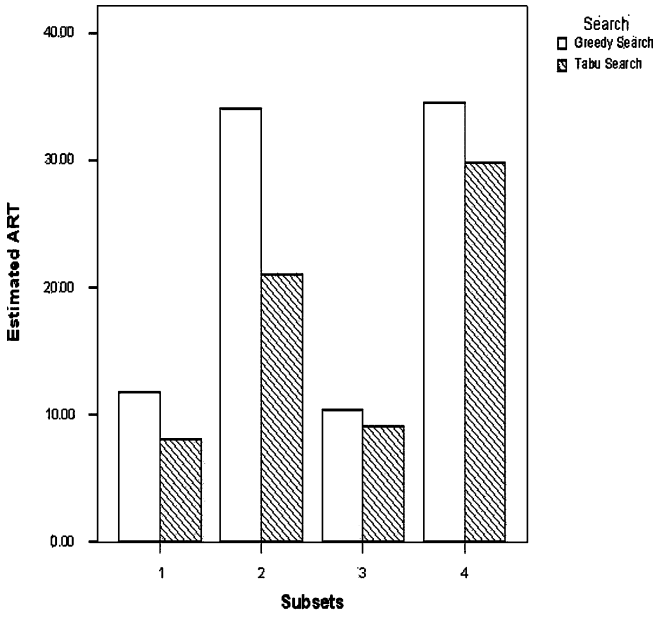
The proposed design framework has been implemented using MATLAB 6.5 and a discrete-event simulator. In order to analyze the design environment, we considered a large-scale Web application that includes 2000 components and 500 pages. The number of components per page is distributed uniformly between three and ten. The application is organized into five site views. The number of pages in each site view is distributed uniformly in the interval [30, 60]. In addition, the distributed environment includes ten locations. The distance between each pair of replica servers is also considered to distribute uniformly in the interval [10, 100]. The distance between a replica server and the origin server is generated randomly as $10u$, where u is a uniform variate in the interval [10, 100]. Further, the number of users at each location is considered to be uniformly distributed in the interval [1000, 5000]. The segment composition of the users was randomly generated.

The interarrival time between requests follows a Pareto distribution with the scale parameter equal to 10 s and the shape parameter equal to 20 [51], [52]. The routing delay between two servers is exponentially distributed with a mean equal to the distance between them. The service time at the replica servers is considered to have a triangular distribution with mean: 0.5 s, mode: 1 s, and max: 1.5 s. The service time of the origin server is also modeled as a triangular distribution with mean: 0.1 s, mode: 0.5 s, and max: 1 s. The request stream is generated based on the distribution of users and their profiles. The write requests constituted 30% of the total number of requests, and only the content at the original server is changed by the write requests.

A fully crossed statistical experimental design consisting of the different levels of the parameters, as shown in Table X, has been used in the study. A total of 72 configurations are generated. For each configuration, ten simulations are performed. (In our following analyses, we mainly concern ourselves with the mean values of the performance metrics. The confidence intervals for those metrics in each configuration are available upon request.) With such a full-crossed experimental design, these configurations are well-dispersed over the design space. The neural network trained by the simulation results of these configurations can well approximate the behavior of the system globally. To examine the direct impact of certain configuration parameters or algorithms, we group the observations into subsets based on the values of the configuration parameters in order to perform ANOVA tests.

TABLE X
Parameter Settings for Simulations

Parameter	Value
Number of Component Groups (k)	10, 30
Content Maintenance Budget (b)	1000, 3000
Number of Network Cells(n)	3, 10
Strong Consistency Threshold (t)	3, 10, 20
Time Interval for Updating (t)	1000, 5000, 10000



Note: Subsets 1: k=10, b=3000; 2: k=10, b=1000; 3: k=30, b=3000; 4: k=30, b=1000.

Fig. 5. Estimated average response time $\hat{y}_1$ by greedy-based search and Tabu search.

B. Content Placement Strategy

In this section, we discuss the content placement strategy as it relates to our attempt to find a better solution. We compare the solution obtained by using Tabu search component placement algorithm with the solution using the greedy algorithm. We vary the number of component groups (k) and the updating cost budget (b) to obtain different configurations. Fig. 5 shows the estimated average response time for four different configurations obtained by the two component placement strategies (Tabu search and greedy algorithm). We find that, in general, the Tabu search algorithm results in at least a 12% improvement over the greedy algorithm in the estimated average response time $\hat{y}_1$. The subsets with smaller numbers of component groups (k) (subsets 1 and 2 in Fig. 5 where $k = 10$) have larger improvement space than the subsets with larger k (subsets 3 and 4, where $k = 30$). Larger numbers of component groups (k) result

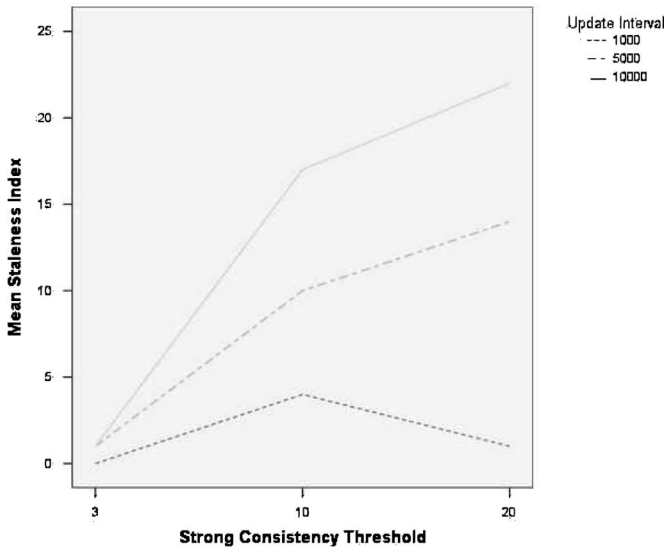


Fig. 6. Staleness index y_5 in the design configuration ($b = 3000, n = 3$, and $k = 10$).

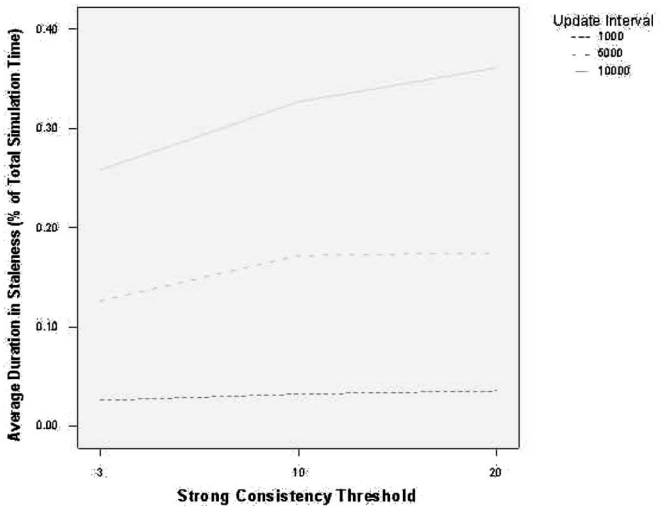
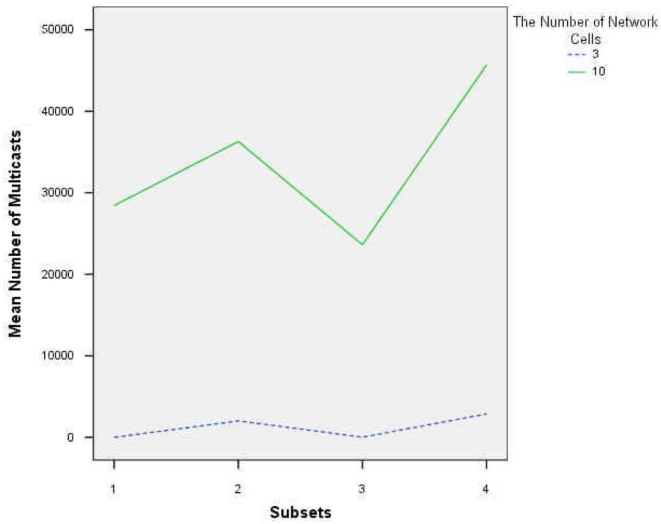


Fig. 7. Average duration of a component in staleness in the design configuration ($b = 3000, n = 3$, and $k = 10$).



Note: Subsets 1: k=10, b=3000; 2: k=10, b=1000; 3: k=30, b=3000; 4: k=30, b=1000.

Fig. 8. Number of multicasts y_3 and the number of network cells n in the design configration ($t = 1000$ and $\tau = 3$).

in smaller component group sizes. Although larger k involves additional computational costs, it is more flexible within the greedy search strategy for group placement thus leaving smaller space for Tabu search to improve the results. The estimated updating cost $\hat{y}_2$ is almost the same in the solutions by the two algorithms for same configurations.

C. Stateless Index

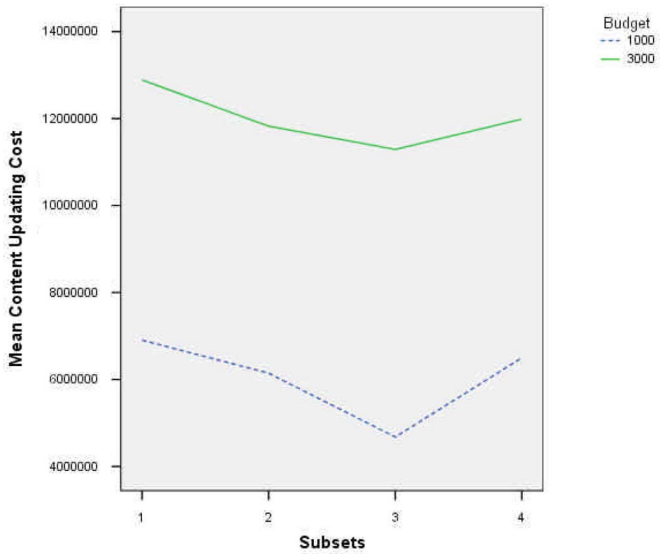
The staleness index (y_5) is an important performance metric that measures the currency of content delivered as the ratio of obsolete content delivered to the total number of requests. The relationship between the staleness index y_5 and the strong consistency threshold for the design configuration ($b = 3000, n = 3$, and $k = 10$) is shown in Fig. 6. We see similar results for other configurations. The results of the experiments show that the staleness index increases almost linearly with the increase of update interval t and the strong consistency threshold (τ). ANOVA tests show that the strong consistency threshold τ and the update interval t have a direct impact on the staleness index, respectively (with p-value $< .01$). Further, Fig. 6 also shows a significant interaction effect between τ and t on the staleness index y_5. This implies that it is possible to achieve the desired level of staleness by appropriately selecting τ and t. Other parameters do not present a significant direct impact on the staleness index.

Another interesting concern is the question of how long a component will stay in the inconsistent state (we called it duration in staleness) given certain design configurations. Fig. 7 shows the average duration of a component in staleness as a function of strong consistency threshold for the configuration ($b = 3000, n = 3$, and $k = 10$). Similar results were observed for other design configurations. It varies apparently with the change of the strong consistency threshold τ and the update interval t. ANOVA tests show that they have a direct impact on the average duration in staleness, respectively (with p-value $< .01$), while other parameters do not have.

As we can expect, smaller τ or t increases the frequency of update, which results in a smaller staleness index and a shorter average duration of a component in staleness.

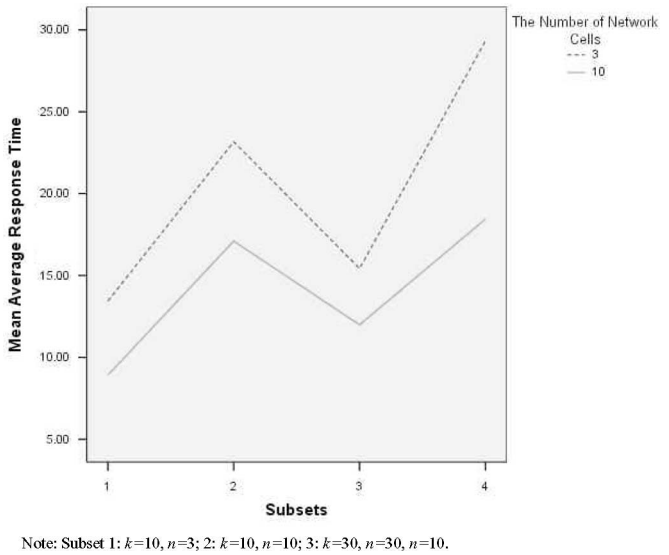
D. Number of Multicasts for Request Routing

Multicasts are used to determine the location of content. In order to measure the traffic generated by request routing, we count the number of multicast messages in a simulation period. Fig. 8 shows the number of multicast messages for request routing (y_3) for various parameter settings (indicated by subsets 1–4 on the x-axis) and its effect on the number of network cells (n) given an update interval of $t = 1000$ and strong consistency threshold of 3. Similar results were observed for the update intervals of $t =$ 10000, 5000, and 1000 and the strong consistency threshold of $\tau = 3, 10,$ and 20. The larger n always introduces a larger number of multicasts. An ANOVA test shows that n has a significant



Note: Subsets 1: k=10, n=3; 2: k=10, n=10; 3: k=30, n=3; 4: k=30, n=10.

Fig. 9. Content updating cost y_2 and the updating cost budget b in the design configuration ($t = 1000$ and $\tau = 3$).



Note: Subset 1: k=10, n=3; 2: k=10, n=10; 3: k=30, n=30, n=10.

Fig. 10. Average response time y_1 and the updating cost budget b in the design configuration ($t = 1000$ and $\tau = 3$).

direct impact on the number of multicasts (with p-value $< .01$). Larger n results in smaller size of network cells and, consequently, more components are located through multicasts.

Several simulation experiments were conducted to study the impact of one's budget on the number of multicasts for different design configurations. An ANOVA test shows that b has a significant direct impact on the number of multicasts (with p-value $< .01$). The simulation experiments show that smaller budgets lead to fewer components being placed in a local edge server and this results in a greater number of components being located through multicasts.

Simulation experiments were also conducted to determine the number of multicasts as a function of the number of component groups. The experiments included several experiments with varying parameter settings. We did not find a significant direct impact that k has on the number of multicasts (with p-value $< .01$). Nor did we find evidence of a significant direct impact on the strong consistency threshold τ and the update interval t for the number of multicasts in the current configuration ranges.

E. Content Updating Cost

In a typical database operation we are worried about content updating cost and the response time for a query. In this section, we focus on updating cost while, in the next section, we discuss average response time.

The relationship between the content updating cost y_2 and the updating cost budget b is shown in Fig. 9 (for various parameter settings indicated by subsets 1–4 on the x-axis in the figure) given an update interval of $t = 1000$ and strong consistency threshold of 3. Simulation experiments varying the update interval $(t = 10\,000, 5000, 1000)$ and the strong consistency threshold ($\tau = 3, 10, 20$) show similar results. An ANOVA test shows the significant direct impact of b on y_2 (with p-value $< .01$). This observation is consistent with our earlier findings. A higher updating budget will introduce more components to be placed in local edge servers and, consequently, a higher consistency cost is needed.

Several simulation experiments were conducted to study the impact of content updating cost y_2 and the number of network cells n in current configuration ranges. We repeated this experiment for several subsets of parameters and by varying the update interval $(t = 10000, 5000, 1000)$ and the strong consistency threshold ($\tau = 3, 10, 20$) showing similar results. No clear relationship could be established between content updating cost and the number of network cells.

The smaller component group size is expected to more effectively utilize the updating budget. Simulation experiments were also conducted varying the update interval $(t = 10000, 5000, 1000)$ and the strong consistency threshold ($\tau = 3, 10, 20$) for a number of subset configurations to determine the impact of the number of component groups on the updating cost y_2. Results show that the updating cost is not affected by the number of component groups. The updating cost is almost identical for $k = 10$ and $k = 30$. Further, an ANOVA test shows that for the different subsets of configurations with $k = 10$ and $k = 30$, respectively, this is also not significant (with p-value $> .01$). Varying the update interval $(t = 10\,000, 5000, 1000)$ and the strong consistency threshold ($\tau = 3, 10, 20$) for the subsets showed similar results.

F. Average Response Time

In this section, we study how the average response time is affected by the number of network cells, budget, and the number of component groups.

Simulation results show that, in general, a smaller number of network cells n results in larger average response time y_1. This is depicted in Fig. 10, which shows the relationship between the average response time y_1 and the number of network cells n in the design configuration of $t = 1000$ and $\tau = 3$. The pattern is held for other configurations of t and τ. Further, ANOVA

tests show that the direct impact of n on y_1 is significant (with p-value $< .01$). When the number of network cells is larger (for example, $n = 10$), the user requests are usually filled by the local edge server as compared to when there are less network cells (for example, $n = 3$). In this case, on receipt of a user request, the cell metaserver will first check whether any edge server inside the cell hosts the requested component before it requests it from outside. The edge server in the cell that hosts the component may not be the nearest server that has the component as in the case of $n = 10$.

Experimental results also show that larger updating budgets lead to a lower average response time. With a higher updating budget, more components can be fetched from local edge servers and, consequently, this leads to a lower average response time.

Simulation results also show that the number of component groups k has a significant impact on the average response time. Larger k results in smaller group sizes, which brings us the flexibility in the content placement to achieve the optimality. Consequently, it results in a smaller average response time.

Simulation experiments show that the average response time is not affected by the strong consistency threshold τ and the update interval t.

G. Feedback and Learning Systems

Through the cross validation, a two-layer feedforward neural network with ten neurons in the first layer and four neurons in the second layer is employed for learning and feedback. The 72 observations are divided into training, validation, and test subsets. One-fourth of the data is used for the validation set, one-fourth for the test set, and one-half for the training set. The sets are picked as equally spaced points throughout the original data. Early-stopping is used in network training. The training stopped after 15 iterations because the validation error increased. This result is reasonable, since the test set error and the validation set error have similar characteristics, and no significant overfitting seems to have occurred.

Subsequently, we performed an analysis of the network response. The entire data set (training, validation, and test) is put through the network and a linear regression between the network outputs and the corresponding observations is performed. Four outputs are obtained, and, consequently, four regressions were performed. The results show significant correlations with $R^2 > .92$ for all regressions. As a result, the trained neural network is able to predict the performance results of other system configurations. With the trained neural network, we can explore the decision space defined by the control variables including the number of component groups (k), the updating cost budget (b), the number of network cells (n), the strong consistency threshold (τ), and the time interval for updating (t). For those configurations that have good predicted results, we may run through the optimization and controlled simulation process. Further, the newly obtained simulation results can be used to retrain the neural network. We may repeat the process until the neural network converges. However, as distinct from linear or nonlinear regressions, the neural network is more of a "black box." We cannot obtain the coefficients for a meaningful interpretation.

VI. Conclusion

In this paper, a feedback learning design framework is developed for shared content management that uses problem decomposition and controlled simulation. The design framework developed in this paper combines the structure of content and service components to design effective replica hosting architectures. A large set of stochastic design parameters with conflicting yet closely coupled design objectives is considered. The design framework is tested using a case study. It can be an effective decision support tool for a system designer to systematically explore design options and select an appropriate design configuration that best meets the design objectives.

Including server service time as a factor in the design framework is a proposed future enhancement to the framework. The framework itself could be modified wherein subproblem 1 and subproblem 2 could be executed in cyclical fashion until an optimal solution of the number of component groups (k) is reached. This, in fact, would result in another solution mechanism in which cell formation and guided simulation could be pursued after the optimal number of component groups is determined. In this scenario, the number of component groups k is a local control variable only for content placement, while, in the framework we presented, the number of component groups k is a global control variable. There are two main benefits of viewing k as a global control variable: 1) k is locally optimal for the content placement. However, it may not be globally optimal for the system when we incorporate the issues of network architecture and content consistency mechanisms and 2), viewing k as a global control variable helps us take account of the interaction effects between other configuration parameters. Contrasting how the solutions would differ if we were to compare the two ways of structuring the solution would be an interesting future extension.

Component hierarchy is an important issue. However, in the current model, we consider content structure to be an exogenous variable for our replica hosting system design. We assume that the Website or the content structure has already been constructed prior to content replication. The purpose of the hierarchical model of content architecture is to describe the structure of the content and characterize the user's behavior. Content could also have been modeled by using Markov chain to capture the interdependencies of the pages. Interoperating navigation patterns in content access and, consequently, the content distribution and reassembling strategies can also be considered.

Further extensions can include the use of an enhanced version of the metric average response time. We have assumed that the average latency is adequate for the purposes of the modeling. This is a limitation of the model and incorporating the standard deviation as a measure would improve the metric and, hence, the solutions. The proposed framework could also be extended by considering different monetary costs of content delivery for servers at different locations. How the reliability of replica servers and the time variation of traffic pattern affect the system performance and design is another interesting topic.

REFERENCES

[1] A. Datta, K. Dutta, H. Thomas, and D. VanderMeer, "World Wide Wait: A study of Internet scalability and cache-based approaches to alleviate it," *Manage. Sci.*, vol. 49, pp. 1425–1444, Oct. 2003.

[2] Zona Research Inc., *The Economic Impacts of Unacceptable Web Site Download Speeds*. Redwood City, CA: Zona Research, 1999.

[3] D. F. Galletta, R. Henry, S. McCoy, and P. Polak, "Web site delays: How tolerant are users?," *J. Assoc. Inf. Syst.*, vol. 5, pp. 1–28, Jan. 2004.

[4] D. Zeng, F.-Y. Wang, and M. Liu, "Efficient Web content delivery using proxy caching techniques," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 34, no. 3, pp. 270–280, Aug. 2004.

[5] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Boston, MA: Addison-Wesley, 2002.

[6] S. Hull, *Content Delivery Networks*. New York: McGraw-Hill, 2002.

[7] D. C. Verma, *Content Distribution Networks: An Engineering Approach*. New York: Wiley, 2002.

[8] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Comput.*, vol. 6, no. 5, pp. 50–85, Sep./Oct. 2002.

[9] M. Rabinovich and A. Aggarwal, "Radar: A scalable architecture for a global Web hosting service," *Comput. Netw.*, vol. 31, pp. 1545–1561, 1999.

[10] P. Rodriguez and S. Sibal, "SPREAD: Scalable platform for reliable and efficient automated distribution," *Comput. Netw.*, vol. 33, pp. 33–49, 2000.

[11] M. Day, B. Cain, G. Tomlinson, and P. Rzewski, "A model for content internetworking," The Internet Society [Request for Comments], Reston, VA, Rep. RFC 3466, 2003.

[12] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. V. Steen, "Replication for Web hosting systems," *ACM Comput. Surv.*, vol. 36, pp. 291–334, Sep. 2004.

[13] A. Umar. (2003). [Online]. *E-Business and Distributed Systems Handbook: Architecture Module*. Nge Solutions. Available: www.ngesolutions.com.

[14] Akamai Technologies, Inc. and Oracle Corporation Edge Side Includes. (2007, Oct. 1). [Online]. Available: http://www.akamai.com/html/support/esi.html.

[15] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera, *Designing Data-Intensive Web Applications*. San Francisco, CA: Margan Kaufmann, 2003.

[16] Y. C. Ho, C. G. Cassandras, C. H. Chen, and L. Y. Dai, "Ordinal optimization and simulation," *J. Oper. Res. Soc.*, vol. 51, pp. 490–500, 2000.

[17] S. Bhattacharjee, R. Ramesh, and S. Zionts, "A design framework for e-business infrastructure integration and resource management," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 31, no. 3, pp. 304–319, Aug. 2001.

[18] F. Glover, J. P. Kelly, and M. Laguna, "New advances for wedding optimization and simulation," in *Proc. 1999 Winter Simul. Conf.*, 1999, pp. 255–260.

[19] D. Duvivier, V. Dhaevers, V. Bachelet, and A. Artiba, "Integrating simulation and optimization of manufacturing systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 33, no. 2, pp. 186–192, May 2003.

[20] B. N. Shao and H. R. Rao, "A comparative analysis of information acquisition mechanisms for discrete resource allocation," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 31, no. 3, pp. 199–209, May 2001.

[21] J. P. Shim, M. Warkentin, J. F. Courtney, D. J. Power, R. Sharda, and C. Carlsson, "Past, present, and future of decision support technology," *Decis. Support Syst.*, vol. 33, pp. 111–126, 2002.

[22] R. H. Kewley and M. J. Embrechts, "Computational military tactical planning system," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 32, no. 2, pp. 161–171, May 2002.

[23] G. Peng, "CDN: Content distribution network," Exp. Comput. Syst. Lab., State Univ. New York, Stony Brook, NY, Tech. Rep. TR-125, Nov. 22, 2004.

[24] D. Olshefski, J. Nieh, and D. Agrawal, "Using certes to infer client response time at the Web server," *ACM Trans. Comput. Syst.*, vol. 22, pp. 49–93, 2004.

[25] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-based scheduling to improve Web performance," *ACM Trans. Comput. Syst.*, vol. 21, pp. 207–233, 2003.

[26] B. Huffaker, M. Fomenkov, D. J. Plummer, D. Moore, and K. Claffy, "Distance metrics in the Internet," presented at the Int. Telecommun. Symp., Natal, RN, Brazil, 2002.

[27] G. Pierre, M. Van Steen, and A. Tanenbaum, "Dynamically selecting optimal distribution strategies for Web documents," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 637–651, Jun. 2002.

[28] M. J. Janiga, G. Dilbner, and F. J. Governali, *Internet Infrastructure: Content delivery*. New York: Goldman Sachs Global Equity Research, 2001.

[29] M. Sayal, P. Sheuermann, and R. Vingralek, "Content replication in Web++," in *Proc. 2nd Int. Symp. Netw. Comput. Appl.*, Cambridge, MA, 2003, pp. 33–40.

[30] B. Li, M. J. Golin, G. F. Italiano, and X. Anddeng, "On the optimal placement of Web proxies in the Internet," in *Proc. 18th INFOCOM Conf.*, New York, 1999, pp. 1282–1290.

[31] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of Web server replicas," in *Proc. 20th INFOCOM Conf.*, Anchorage, AK, 2001, pp. 1587–1596.

[32] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz, "Clustering Web content for efficient replication," in *Proc. 10th IEEE Int. Conf. Netw. Protocols (ICNP 2002)*, Los Alamitos, CA, Nov., pp. 165–174.

[33] J. Kangasharju, J. Roberts, and K. Ross, "Object replication strategies in content distribution networks," presented at the 6th Web Caching Workshop, Boston, MA, 2001.

[34] O. Ardaiz, F. Freitag, and L. Navarro, "Improving the service time of Web clients using server redirection," presented at the 2nd Workshop on Perform. Archit. Web Serv., Cambridge, MA, 2001.

[35] M. Szymaniak, G. Pierre, and M. Van Steen, "Netairt: A DNS-based redirection system for apache," presented at the Int. Conf. WWW/Internet, Algarve, Portugal, 2003.

[36] V. Cardellini, M. Colajanni, and P. S. Yu, "Request redirection algorithms for distributed Web systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 4, pp. 355–368, Apr. 2003.

[37] V. Cate, "Alex—a global file system," in *Proc. File Syst. Workshop*, Ann Harbor, MI, 1992, pp. 1–11.

[38] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive push-pull: Disseminating dynamic Web data," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 652–668, Jun. 2002.

[39] N. Krishnakumar and A. J. Bernstein, "Bounded ignorance: A technique for increasing concurrency in a replicated system," *ACM Trans. Database Syst.*, vol. 4, pp. 586–625, 1994.

[40] C. Gray and D. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," in *Proc. 12th Symp. Oper. Syst. Princ.*, Litchfield Park, AZ, 1989, pp. 202–210.

[41] M. Wedel and W. A. Kamakura, *Market Segmentation: Conceptual and Methodological Foundations*. Boston, MA: Kluwer Academic, 1998.

[42] G. Holland and K. Kumar, "Component-based Web page composition," in *Proc. 6th Int. Conf. Object-Oriented Inf. Syst.*, London, U.K., Dec. 2000, pp. 177–199.

[43] K. Dutta, A. Datta, H. Thomas, and P. Keskinocak, *Optimizing Caching in Object-Oriented Applications*. Pittsburgh, PA: Carnegie, 2002.

[44] F. J. Torres-Rojas, M. Ahamad, and M. Raynal, "Timed consistency for shared distributed objects," in *Proc. 18th Symp. Principles Distrib. Comput.*, Atlanta, GA, 1999, pp. 163–172.

[45] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ: Prentice-Hall, 2002.

[46] B. S. Everitt, S. Landau, and M. Leese, *Cluster Analysis*, 4th ed. London, U.K.: Edward Arnold, 2001.

[47] F. Glover and T. Laguna, *Tabu Search*. Dordrecht, The Netherlands: Kluwer Academic, 1997.

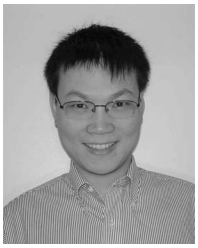
[48] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2003.

[49] M. T. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[50] H. Demuth and M. Beale, *Neural Network Toolbox for Use with Matlab, User's Guide*. Natick, MA: The MathWorks, Inc., 2002.

[51] M. J. Fischer, A. M. Girard, D. M. Masi, and D. Gross, "Simulating Internet-type queues with heavy-tailed service or interarrival times," presented at the Appl. Telecommun. Symp., 2001 Adv. Simul. Technol. Conf., Seattle, WA.

[52] F. Hernandez-Campos, J. S. Marron, G. Samorodnitsky, and F. D. Smith, "Variable heavy tailed durations in Internet traffic, Part I: Understanding heavy tails," in *Proc. ACM/IEEE MASCOTS 2002*, pp. 43–50.



Jingguo Wang received the B. S. degree in computer science from Fudan University, Shanghai, China, in 1998, and the M.S. degree in operations research and the Ph.D. degree in management science and systems from the State University of New York, Buffalo, in 2005 and 2007, respectively.

Currently, he is an Assistant Professor in the College of Business Administration, University of Texas, Arlington. He is the author or coauthor of several research papers published in international journals such as the *European Journal of Operational Research* and the *Journal of Multi-Criteria Decision Analysis*. His current research interests include distributed computing, information assurance, and decision making.

Dr. Wang has been a participant at conferences such as the International Conference on Information Systems and the Americas' Conference on Information Systems (AMCIS). He was the recipient of the Best Paper Award at AMCIS 2006.

Raj Sharman (S'96–A'97–M'05) received the B.Tech. and M.Tech. degrees from IIT Bombay, Bombay, India, in 1980 and 1983, respectively, and the M.S degree in industrial engineering and the Ph.D. degree in computer science from Louisiana State University, Baton Rouge, in 1987 and 1998, respectively.

Currently, he is an Assistant Professor in the School of Management, State University of New York, Buffalo. His current research interests include distributed computing, decision support systems, information assurance, and disaster response management.

Ram Ramesh (A'99–M'04) received the B.Tech., M.Tech., and Ph.D. degrees in 1975, 1977, and 1985, respectively.

He is currently a Professor in the School of Management, State University of New York, Buffalo. He is the author or coauthor of several research papers published in international journals such as: the *INFORMS Journal on Computing*, *Information Systems Research, Naval Research Logistics, Management Science*, and the *Communications of the ACM*. His current research interests include conceptual modeling (ontologies, connectionist modeling, and nonmonotonic reasoning), economics and technologies of Internet capacity provision networks, and database systems and distributed computing frameworks. His research has been funded by several Department of Defense organizations and contractors including the Army Research Institute, the Air Force Office of Scientific Research, the Naval Training Systems Center, Westinghouse, Raytheon, and Samsung. He currently serves as an Associate Editor for the *INFORMS Journal on Computing*, the *Communications of the AIS*, the *Journal of Semantic Web and Information Systems*, and the *Journal of Intelligent Information Technologies*. He is also a Co-Editor-in-Chief of the *Information Systems Frontiers*. He is a Guest Editor of the *Journal of AIS*.