Multimedia Systems 9: 398–410 (2004)
Digital Object Identifier (DOI) 10.1007/s00530-003-0113-4

Multimedia Systems

Directory-based composite routing and scheduling policies for dynamic multimedia environments

Zhenghua Fu, Nalini Venkatasubramanian

Department of Information and Computer Science, University of California at Irvine, Irvine, CA 92697-3425, USA
e-mail: {zfu|nalini}@ics.uci.edu

Abstract. In this paper we present and evaluate algorithms to address combined path and server selection (CPSS) problems in highly dynamic multimedia environments. Our goal is to ensure effective utilization of network and server resources while tolerating imprecision in system state information. Components within the framework implement the optimized scheduling policies as well as collect/update the network and server parameters using a directory service. We present and analyze multiple policies to solve the CPSS problem. In addition, we study multiple techniques for updating the directory service with system state information. We further evaluate the performance of the CPSS policies under different update mechanisms and study the implications of the CPSS policies on directory service management.

1. Introduction

Advances in computation, storage, and communication technologies are initiating the large-scale deployment of multimedia services and applications such as distance learning, video-on-demand, multimedia conferencing, video phones, and multiparty games. The evolution of the Internet and differentiated services has also expanded the scope of the global information infrastructure and increased connectivity among service providers and clients requesting services for multimedia applications. As this infrastructure scales, service providers will need to replicate data and resources on the network to serve more concurrent clients. Efficient and adaptive resource management mechanisms are required to deal with highly dynamic environments (e.g., those that involve mobile clients and hosts) to ensure effective utilization of resources while supporting increasing numbers of requests. Multimedia applications require Quality of Service (QoS) guarantees; resource provisioning techniques for multimedia applications must ensure that QoS requirements are met both at the server and along the network path.

The QoS-based resource provisioning problem has been addressed independently at multiple levels. QoS routing techniques have been proposed to improve the network utilization by balancing the load among the individual network links. Server selection and load balancing policies at the middleware layer direct the user to the "best" server while statically treating the network path leading from the client to the server as predetermined by the routing tables, even though there may exist multiple alternative paths. While the two techniques can independently achieve some degree of load balancing, we argue that in multimedia environments, where applications are highly sensitive to QoS parameters like bandwidth and delay, high-level provisioning mechanisms are required to address the route selection and server selection problem in a unified way. Such integrated mechanisms can potentially achieve higher systemwide utilization and therefore allow more concurrent users.

In the future we can expect highly dynamic network topologies where heterogeneous wired and wireless networks provide access to an increasing number of mobile users. Optimizing resource utilization becomes further complicated in this case. Typically, resource provisioning algorithms rely on accurate information about current resource availabilities. In a highly dynamic and ad hoc environment where clients are mobile, cost-effective QoS provisioning techniques (e.g., load-sensitive routing and scheduling) must be able to tolerate information imprecision and work effectively with approximate system state information. In such an environment, the information collection and scheduling processes must cooperate with each other; they cannot be viewed as completely independent components in the QoS provisioning architecture. In this paper we develop a framework in which scheduling decisions for a client request are based on path as well as server qualities. The middleware approach in this paper illustrates the use of a directory service that serves as an information repository whose use makes it possible to effectively tackle the combined path and server selection (CPSS) problem. Specifically, we

(a) develop a model and algorithm to address the CPSS problem based on system residue capacities [FV99],
(b) develop and evaluate a family of directory-enabled policies for composite routing and scheduling [19], and
(c) test and understand the performance of the CPSS policies under varying traffic patterns and under different levels of information imprecision in the directory service.

The rest of this paper is organized as follows. Section 2 describes the server and network model and presents the general

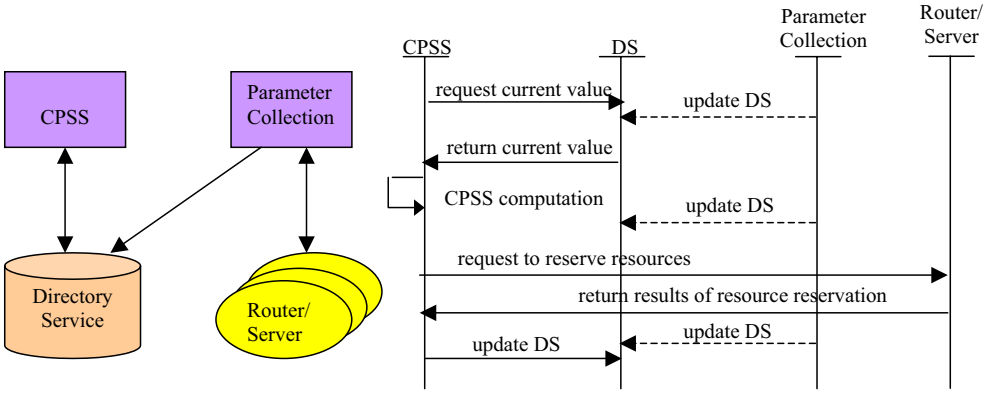


Fig. 1. System architecture and operational flow of the CPSS process

case for the CPSS algorithm. In Sect. 3 we develop a family of CPSS policies that address the CPSS problem effectively under dynamically varying system and network conditions. Section 4 deals with the collection and update of state information using a directory service and discusses several policies for information update. Section 5 contains a performance evaluation of the CPSS policies and update mechanisms. Section 6 describes related work, and Sect. 7 concludes with future research directions.

2. Combined path and server selection (CPSS)

Figure 1 illustrates the overall architecture and operational flow of the CPSS process. The crux of the architecture is a directory service (DS) that maintains information about the current system state that includes (a) topologies of server and network interconnectivity, (b) replica information, and (c) server and network resource availabilities. The DS information is used by the CPSS module to perform resource provisioning and maintained by the parameter collection and update process (Sect. 4).

The general flow of the CPSS process is as follows. A request containing QoS parameters is initiated at a source node; a directory service provides the required system information and makes path and server assignments for the client request. Given the assignment, the client node proceeds to set up a connection along the assigned network path to the server. The routes and the servers check their residue capacity and either admit the connection by reserving resources or reject the request. When the connection terminates, the client sends the termination requests and the resources are reclaimed along the connection.

2.1. Modeling the CPSS problem

In this section, we briefly formulate the CPSS problem and develop an algorithm for addressing path and server selection in a unified manner. We model the QoS requirement of the request R from client c as a triple: the path, the server, and the end-to-end quality:

$$R :< PATH_R, SERV_R, ETOE_R > \ .$$

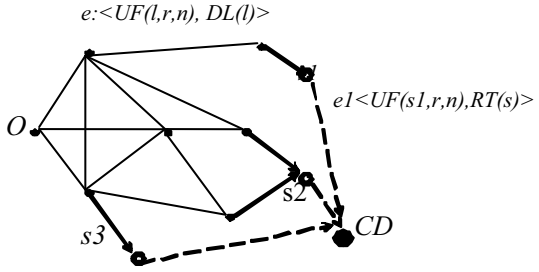


Fig. 2. CPSS diagram graph $G :< N, E >$ with the client request at point O and a set of target servers $S : s1, s2, s3$. G is extended by adding a point CD and artificial edges for $s1$, $s2$, and $s3$

The path QoS requirement is defined as the bandwidth required for the request R along any chosen path:

$$PATH_R :< BW_R >$$

The server QoS requirement is spelled out in terms of the server resources required by the request R. The capacity of a multimedia server can be specified as three equally important bottleneck parameters: CPU cycles, memory buffers, and I/O bandwidth:

$$SERV_R :< CPU_R, \quad BUF_R, DB_R > \ .$$

The end-to-end requirement $ETOE_R$ is quantified as the end-to-end delay DL_R tolerated by the request R:

$$ETOE_R :< DL_R > \ .$$

We now model the system as a directed graph $G < N, E >$, where the distributed servers are represented as nodes connected to one or more router nodes (Fig. 2). A directed edge from a router to a server is used to represent each connection to a server in graph G. For a link l we use a term BW^l_{avail} to note its available bandwidth and term DL^l to denote its current delay (the delay includes the propagation delay and the queuing delay at the transmit end). By definition, for a path p we have

$$BW^p_{avail} = \min_{l \in p} \{BW^l_{avail}\}; \quad DL^p = \sum_{l \in p} DL^l \ .$$

As can be observed, the bandwidth parameter is a bottleneck factor and the delay parameter is an additive factor.

If we use RT^s to denote the response time of a server s, then given an assignment $X = \{p, s\}$ with network path p and server s, the end-to-end delay of assignment X, EED^X, is the sum of the link delays and the response time of the server:

$$EED^X = DL^p + RT^s, p, s \in X \ .$$

To deal with path and server selection in a unified way, we first define a utilization factor for network links and servers to quantify the residue capacity and then proceed to define a *distance* between the client and a server. The utilization factors for a link l, given a request r and a parameter n, is defined as

$$\begin{cases} UF(l,r,n) = \left(\dfrac{1}{BW^l_{avail} - BW_r}\right)^n \ , \\ UF(l,r,n) = \infty, if\ otherwise; \\ \quad \text{if} \quad BW^l_{avail} > BW_r \ . \end{cases}$$

The utilization factor for a server s, given a request r and a parameter n, is defined as

$$\begin{cases} UF(s,r,n) = \left(\text{Max}\left(\dfrac{1}{CPU^s_{avail} - CPU_r}, \right.\right. \\ \qquad \left.\left. \dfrac{1}{MEM^s_{avail} - MEM_r}, \dfrac{1}{DB^s_{avail} - DB_r}\right)\right)^n \\ \text{if available capacities greater than requested} \\ UF(s,r,n) = \infty, \quad \text{otherwise.} \end{cases}$$

In the above equations, the reciprocal of the residual capacity indicates the impact of request r on link l (or server s). In the case of the server utilization factor, $UF(s,r,n)$, the bottleneck resource (i.e., one that is impacted the most) is conservatively used to estimate the impact of the request r on server s. The parameter n in the utilization factor represents the degree to which a lightly loaded resource is favored over a congested resource [28]. It serves as a tuning knob to bias resource allocation in favor of lightly loaded resources. For an assignment $X = \{p, s\}$, we define the distance of the server s to be

$$\text{Dist}(s,r,n) = \sum_{l \in p, p \in X} UF(l,r,n) + UF(s,r,n), \ \ s \in X \ .$$

The feasibility condition. Given a client request $R :< BW_R, CPU_R, MEM_R, DB_R, DL_R >$, an assignment $X = \{p, s\}$ is feasible if and only if it satisfies all the following:

- $BW^{p*}_{avail} \geq BW_R$;
- $CPU^{s*}_{avail} \geq CPU_R, BUF^{s*}_{avail} \geq BUF_R, DB^{s*}_{avail} \geq DB_R$;
- $EED^{X*} \leq DL_R$.

We define a feasible set X_f as a set of all the assignments that meet the feasibility condition.

Optimality of CPSS. Given a client request $R :< BW_R, CPU_R, BUF_R, DB_R, DL_R >$, an assignment $X^* = \{p^*, s^*\}$ is optimal if and only if it satisfies the feasibility condition and a policy-dependent optimality criterion. For instance, the optimality clause for the BEST UF policy is

$$\text{Dist}(s^*, r, n) = \text{Min}\{\text{Dist}(s, r, n)\}, \quad \text{for all } s \text{ in feasible set } S \ .$$

We will discuss specific CPSS policies in detail later in Sect. 3.

2.2. The CPSS algorithm

In this section we present an algorithm to solve the CPSS problem. Given a network topology G, a client request from point O, and a target set S of replicated servers that contain the information and service requested by the client, we extend the existing topology $G < N, E >$ to $G' < N', E' >$ by adding one node called common destination, CD, to graph G and one artificial edge per server s in target set S, denoted e_s, $s \in S$ from the server s to the common destination CD, (see Fig. 2 above).

The weight of e is defined as $W(e) :< UF, DL >$, the two additive parameters representing the load level and the delay, respectively. Specifically, the weight function W in G' is derived as follows:

(a) for edge $e(u,v)$ in E, define $W(e) =< UF(e,r,n), DL^e >$;
(b) for edge $e'(s, CD)$ not in E, but in E', define $W(e') =< UF(s,r,n), RT^s >$.

To simplify the graph, we remove from G' those edges in which the available capacity is less than that requested. We calculate a set of paths from the origin O to CD to form a feasible assignment set X_f, subject to the end-to-end delay constraint. In the appendix, we prove the feasibility conditions for such an assignment derived from a path P.

Now, we want to find a path with maximum utility factor value while satisfying the delay constraint. This problem can be cast as a restricted shortest path (RSP) problem with both a bottleneck parameter (utility factor) and an additive parameter (delay constraint). Although it has been proved that the general case of such an RSP problem is NP-hard [23], there exist heuristic techniques (e.g., dynamic programming) to solve it by assuming an integer value of the delay constraint [23,29,8]. The dynamic programming approach can be summarized as follows. Basically, for the specific source node, the system establishes a dynamic table of D rows and $|N|-1$ columns, each cell representing the maximum utility factor from source to a certain node (corresponding column) within a delay constraint (corresponding row). The algorithm starts from the neighbor nodes of the source and propagates until the dynamic table is filled up. After the algorithm terminates, the column of the destination node represents the maximum utility factor values of various delay constraints in [1,D]. The final answer is then the maximum value within the column. A detailed implementation of the RSP heuristic algorithm is provided in [29]. In this paper, we apply the RSP heuristic in our CPSS algorithm to find a feasible set X_f from the extended graph G'. After the algorithm terminates, a feasible path set can then be derived from the dynamic table. The complexity of this RSP algorithm is $O(D|N|)$.

The CPSS algorithm. $(G' < N', E' >, R, O, n,)$

1. */* initialization */*
   For each edge $e(u, v)$ in G':
   If e is in E,
   if $UF(e, R, , n) = INFINITY$
   delete edge e from G'.
   Else
   $W(e).dist = UF(e, R, n); W(e).delay = DL^e$
   Else */*e is an artificial arc,* $e = (s, CD)$. $*$ /*
   If $UF(s, R, n) = INFINITY$,
   delete edge e from G'.
   Else
   $W(e).dist = UF(s, R, n); W(e).delay = RT^s$.
2. */* Run the RSP algorithm to obtain the feasible path set* X_f:
   $X_f = \{P | P\{(O, v1), (v1, v2), .., (s, CD)\} * /$
   $X_f = RSP(G' < N', E' >, W, O, CD, DLr)$.
3. Calculate optimal assignment $X^* = \{P^* \backslash (s^*, CD)$ $s^*\}$ based on CPSS policy
4. Return X^*.

3. A family of CPSS policies

We propose deterministic and nondeterministic CPSS policies that choose an optimal assignment from X_f. Our objective is to improve the overall system utilization and number of concurrent users; therefore we focus on policies that minimize the usage of system resources while balancing the load across the links and servers.

Deterministic CPSS policies

Shortest hop. Choose an assignment from the feasible set X_f so that the number of hops from source to destination is minimal: $X^* :< p^*, s^* >$ so that $\mathrm{Hop}(X^*) = \mathrm{Min}\{\mathrm{Hop}(X) | .$ For all X in feasible set $X_f\}$. This variation of the traditional shortest-path policy provides a shortest widest solution.

Best UF. Choose an assignment from the feasible set such that the utilization factor (UF) is minimal:

$$X^* :< p^*, s^* > \text{ so that } \mathrm{Dist}(X^*) = UF(p^*) + UF(s^*) = \mathrm{Min}(\mathrm{Dist}(X) | \text{ for all } X \text{ in feasible set } X_f).$$

The best UF policy is a variation of online algorithms that maximize the overall utilization of resources in the system without knowledge of future requests.

Nondeterministic CPSS policies. Nondeterministic policies attempt to achieve a better degree of load balance among the links and servers. We present three nondeterministic CPSS policies: (i) a randomized path selection policy, (ii) a statically weighted differential probabilistic policy, and (iii) a load-sensitive probabilistic policy. The probabilistic policies presented here allow a differential treatment of network and server resources instead of treating the server and network resources in a unified manner. This helps ensure that the more constrained resource can be treated preferentially, yielding better adaptation under dynamic conditions.

Random path selection. Select $X^* :< p^*, s* >$ randomly from the feasible set X_f. By randomly picking a choice from the feasible set, this policy tries to avoid oscillations that are often characteristic of more static policies.

Weighted differential probabilistic policy (Prob-1ϕ). From the feasible set X_f we calculate a selection probability for each Xi based on its residue capacity relative to other assignments. The probability is defined as

$$Sel\_\mathrm{Prob}(X_i) = R_1 \cdot \frac{UF^{-1}(s_i)}{\sum_{X_k \in X} UF^{-1}(s_k)} + R_2 \cdot \frac{UF^{-1}(p_i)}{\sum_{X_k \in X} UF^{-1}(p_k)}.$$

$R1$ and $R2$ decide how the server and path resources should be emphasized in calculating the selection probability of the assignment. For instance, to avoid the situation where we have a good server with a bad path or a good path with a bad server, we can set $R1 = R2 = 0.5$.

Prob-1ϕ (X)
1. From the feasible assignment set $X, X = \{X_1 :< p_1, s_1 >, \ldots, X_n :< p_n, s_n > \}$ calculate distinct server set S, $S = \{s_1, s_2, \ldots, s_k\}, s_i \neq s_j, \forall i, j \in [1, k], i \neq j$.
2. Find the corresponding path set $P = \{p_1, p_2, \ldots, p_k\}$ such that p_i is the shortest path leading to server $s_i, s_i \in S$ and assignment $x :< p_i, s_i > \in X, i = 1, 2, \ldots, k$.
3. The k distinct assignments are then derived as $X' = \{x_1 :< p_1, s_1 >, \ldots, x_k :< p_k, s_k >\}$.
4. Weight X_i in X' as $R_1 \cdot \frac{UF^{-1}(s_i)}{\sum_{x_j \in X'} UF^{-1}(s_j)} + R_2 \cdot \frac{UF^{-1}(p_i)}{\sum_{x_j \in X'} UF^{-1}(p_j)}$.
5. Select an assignment X^* from X' probabilistically.
6. Return the selected assignment X^*.

Load-sensitive differential probabilistic policy (Prob-2ϕ). This load-sensitive algorithm takes into consideration architectural characteristics of servers and network links to yield a two-phase policy that first determines a bottleneck resource. From the feasible set X_f the first phase calculates the average UF values of path and server components to decide which of the two resources constitutes the bottleneck for the client's request.

Prob-2$\phi(X)$
1. From the feasible assignment set X,
$X = \{X_1 :< p_1, s_1 >, \ldots, X_n :< p_n, s_n >\}$ calculate distinct server set S,
$S = \{s_1, s_2, \ldots, s_k\}, s_i \neq s_j, \forall i, j \in [1, k], i \neq j$.
2. From the feasible assignment set X
calculate average UF value of network and server,[1] UF^{NW}_{AVG} and UF^{SVR}_{AVG}.
3. /* Select s^* from server set S according to UF^{NW}_{AVG} and UF^{SVR}_{AVG} :* /
If $UF^{NW}_{AVG}/UF^{SVR}_{AVG} > r, r$ is a threshold parameter.
/* The network is more heavily loaded than the servers */.
Weight all servers in S equally as $\frac{1}{k}$.
Select a server s^* from server set S.
Else /* the servers are more heavily loaded than the network */.
Weight server s_i in S as: $\frac{UF^{-1}(s_i)}{\sum_{j \leq k} UF^{-1}(s_j)}$.
Select a server s^* from S with max weight.
4. /* Select a best path leading to the selected server */.
Determine set $X' \subseteq X$ so that $X' = \{X_i :< p_i, s_i > | s_i = s^*, i = 1, 2, \ldots n\}$.
Weight X_i in X' as $\frac{UF^{-1}(p_i)}{\sum_{j \leq k'} UF^{-1}(p_j)}$.
Select an assignment X^* from X' probabilistically.
5. Return the selected assignment X^*.

Based on the bottleneck factor determined in the first phase, the second phase executes as follows:

(a) If the server resource is determined to be the bottleneck, we emphasize balancing the load between the servers. To do this, we first eliminate multiple paths in X to the same server and build a reduced server set S that contains distinct feasible servers. Next, we probabilistically select a server, s^*, from server set S according to each server's relative UF value in S. Finally, from all the paths leading to that server s^* we probabilistically select a path, p^*, according to relative UF values among all such paths.
(b) If the network is the restricting element, we emphasize balancing the load between the alternative network links. Experiments reveal that many paths in the network share large amounts of network links, so the most effective way of balancing the load between network links is to distribute the request to different servers. Therefore, we select a server s^* first from the server set S using a uniform probability distribution and then probabilistically select a path p^* leading to that s^*. Our goal is to randomize the usage of the network to avoid hot spots and maximize load balance among the various network paths.

4. Directory-enabled parameter collection and update

The CPSS policies discussed are based on a knowledge of network topology, replica maps, and load information of network links and distributed servers. In our framework, this information is maintained in a directory service, i.e., a unified repository to be accessed by a CPSS module for decision making. As an independent module, the directory service holds the following information:

- Network connectivity information (topology),[2]
- An accurate server replica map,[3] and
- Network and server state information (for example, residual link bandwidth, link delay, server capacities).

While the first two types of information may be relatively static, network and server state information changes from time to time and has to be updated periodically in the directory. Obviously, the accuracy of the maintained state information depends on the update frequency, i.e., more frequent updates improve directory accuracy. However, frequent updates introduce additional network traffic and processing power at the routers, servers, and directory service. There is a fundamental cost–accuracy tradeoff that must be addressed in effective directory service maintenance. In this paper, our fundamental objective is to ensure cost-effectiveness of the CPSS process. To this end, we study the information collection techniques together with CPSS policies to directly explore such a tradeoff issue between management cost and effectiveness.

Three factors play a role in determining the efficiency of the overall system where CPSS and information collection policies operate asynchronously: (a) the representation and management of dynamic state information in the directory and the utilization of this information by the CPSS policies, (b) the degree of coupling between the information collection and resource provisioning processes, and (c) the impact of statistical fluctuations in the network and server load. In this paper we evaluate the proposed CPSS policies against a spectrum of directory management strategies at different operating points of the cost–accuracy tradeoff ranging from low-cost scenarios (e.g., infrequent updates of instantaneous values) to more sophisticated policies using a range-based parameter representation. Our goal is to improve the overall request success ratios in low update frequency (low cost) scenarios using our proposed CPSS policies while maintaining high request success ratios if frequent information update is possible. In the following section we discuss two related issues – the representation/use of dynamic state information in the directory and the degree of coupling – and show how the overall proposed framework can address statistical fluctuations in the system.

4.1. Directory representation of dynamic information

We use two techniques to represent dynamic state information in the directory.

(a) *Instantaneous snapshot-based representation*: Here, information about the collected parameter, e.g., residue capacity of network nodes and server nodes, is based on an absolute value obtained from a periodic snapshot. During each update period, probing is initiated to gather the current information of router nodes and server nodes; the directory is subsequently updated with the collected values.
(b) *Range-based representation*: Here, the residue capacity information is maintained in the DS as a range with a

[2] The directory collects this information by participating in routing information exchange.

[3] This is obtained from a distributed domain name service either in an on-demand mode or a caching mode ([15,27]).

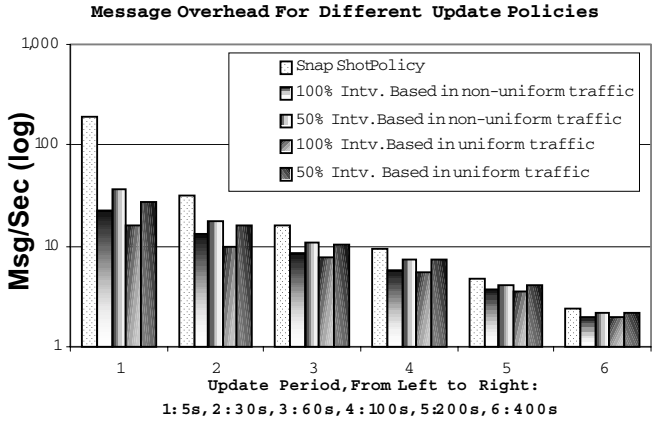


Fig. 3. Message per second for update

lower bound L and an upper bound H; the actual value is assumed to= be uniformly distributed in this range with the expected value $(H-L)/2$. Range-based information collection can have many variations [1,17,18]; our performance evaluation focuses on one of them, i.e., the fixed-interval-based policy. In the fixed-interval-based policy, we divide the entire range of values that a parameter can assume into k fixed-size intervals, each of size B. Instead of representing each parameter with a range $< L, H >$, the residue capacity information is now represented using a range $< kB, (k+1)B >$ with $k \geq 0$. During each update period, a probe is initiated to obtain the current information from router and server nodes. If the current value falls out of the DS range, the directory is updated with another interval-based range; otherwise no update is sent.

The messaging overhead introduced by the information collection/update process can vary significantly based on the data representation and periodicity of update. Figure 3 compares the message overhead cost of three information update methods: instantaneous snapshot, the fixed-interval-based policy with a smaller interval (50% interval-based), and the fixed-interval-based policy with a larger interval (100% larger interval). The y-axis is the number of directory updates in log scale, and the x-axis is the update frequency. We observe that, in general, the instantaneous snapshot-based approach consistently causes more directory updates than the interval-based representation and that the smaller interval has a slightly higher update cost than the larger intervals. Specifically, when the update frequency is high (e.g., with a 1-s period), the message overhead cost for the instantaneous snapshot-based representation is about ten times that with an interval-based representation. When update frequency is very low (e.g., with a 400-s period), the new state information is likely to be different and out of the current interval; therefore, interval-based representations cannot save significantly in terms of directory updates.

CPSS interpretations of the interval. Since a range cannot be used in CPSS calculations, we need to convert a range, say, $[L, H]$, to a number. We further explore three variations.

(a) *Pessimistic (PESS):* uses a lower bound of a uniform distribution corresponding to the current interval.

(b) *Optimistic (OPT):* uses an expected value of a uniform distribution corresponding to the current interval, i.e., $(H-L)/2$.

(c) *Optimistic favor stable (OPT2):* uses an expected value multiplied by a "*degree of stability*" calculated as $\frac{H-L}{Capacity}$.

From our performance studies (see next section) we found that, generally, the OPT2 policy exhibits the best CPSS cost efficiency.

4.2. Degree of coupling between information collection and resource provisioning

The degree of coupling between the information collection and resource provisioning processes plays an important role in addressing the cost–accuracy tradeoffs involved in directory service maintenance.

A model with tight coupling works as follows. Based on information in the DS, the CPSS module determines a suitable path and server assignment for a particular request. The CPSS module informs the DS so that the requested resource is prededucted along the assigned path/server from the state information database prior to the actual resource reservation process. Several negative consequences ensue when the two tasks (information collection and resource reservation) are closely interleaved. Since the CPSS module makes assignments based on approximate state information in the DS, the final success of the request cannot be guaranteed until resources are reserved along the selected path in any case. The underlying network or server may very well reject the request due to lack of resources. Furthermore, the interleaving keeps track of resource occupation (in a somewhat eager fashion) while resource releases (caused by connection termination, failed reservations, etc.) propagate to the DS at a somewhat slower pace. This introduces a biased image of the current system utilization map to new incoming requests by presenting an inaccurate picture of available resources. This is further aggravated in a highly dynamic environment where not all resources are provisioned through a brokerage process.

For these reasons, we do not choose to update resource allocation information in the DS prior to the success of the reservation process. Our hypothesis is that a proactive collection algorithm combined with soft state maintenance is expected to bring further improvement. Therefore, in the following performance evaluation section, the resource state information is only refreshed by periodic updates in the directory service. The CPSS policies make assignment decisions based on this information without changing it. We explore snapshot-based and range-based approaches to independently maintain the DS. In particular, we use static-interval-based collection processes that use fixed size ranges and sampling frequencies in the following performance evaluation section. In [18] we explore more dynamic information collection techniques with variable frequency and adjustable range representation using an autoregressive moving average time-series model. While the more dynamic approach is able to increase the accuracy of the collection process in the directory service under certain conditions, it also incurs high overhead due to the memory required to maintain the moving average model. To focus on

the cost effectiveness of the CPSS policies, we evaluate the relative performance of the CPSS policies with variations of the static-interval-based information collection model.

4.3. The impact of statistical fluctuations

In a real network infrastructure, the statistical fluctuation in the load of network links and servers often leads to inaccuracy of the state information in the directory. Such inaccuracy poses serious challenges for CPSS policies. Since this is an unavoidable issue in practice, we now discuss some design strategies that can help reduce the impact of such fluctuations. In the proposed CPSS framework, *small* statistical fluctuations are accommodated by the range-based information collection techniques, and large, *dramatic* statistical fluctuations are handled jointly by range-based collection and probabilistic CPSS policies. In another paper, we developed a detailed analysis of how the specific statistical distributions (e.g., Poisson or Pareto) affect the accuracies and overhead cost of our range-based information collection technique (the interested reader is referred to [18]). In the following section, we provide a brief description on how both small and dramatic statistical fluctuations are accommodated by the proposed framework.

Small fluctuations in system state. If the system is relatively stable, i.e., the parameter values vary within a small range over time, the fluctuation is filtered out by our range-based sampling approach – the effective outcome of the CPSS policies are not affected. In the range-based approach, we use a *range* to cost effectively approximate the current state parameter instead of a single instantaneous value. There are several strategies that may be used to determine the position and width of the range. In this paper, however, we use fixed ranges to simplify information collection and focus on the discussion of the design and evaluation of CPSS policies. In other work [18], the range is adaptively adjusted (i.e., expanded or contracted) to exploit an optimal tradeoff between information accuracy and sampling overhead using a time-series-based approach.

Large fluctuations in system state. If the system state changes dramatically, the fluctuation is first reflected as a range index jump in the range-based collection policy. This change is then taken into account in our probabilistic CPSS policies, in particular the Prob-2ϕ policy. Prob-2ϕ is a bottleneck-oriented assignment policy that explicitly tries to eliminate hot spots among networks and replica servers and distribute the overall load evenly throughout the system. Specifically, when a dramatic change is observed by the sampling process, the Prob-2ϕ policy first determines whether the network bandwidth or the replica server capacity is the limiting factor for a particular end-to-end request. Accordingly, it chooses an assignment probabilistically from the feasible set calculated by the RSP algorithm to maximize the request/success ratio, thereby avoiding system congestion. Specifically, if the server is the limiting factor, the Prob-2ϕ policy puts priority on distributing the load among the replica servers evenly; if the network bandwidth is the limiting factor, the Prob-2ϕ policy puts priority on avoiding network congestion. In summary, when a dramatic fluctuation is detected, Prob-2ϕ is able to avoid traffic hot spots and congestion by statistically distributing the system load evenly. Thus the user request/success ratio is improved and high overall system utility is achieved.

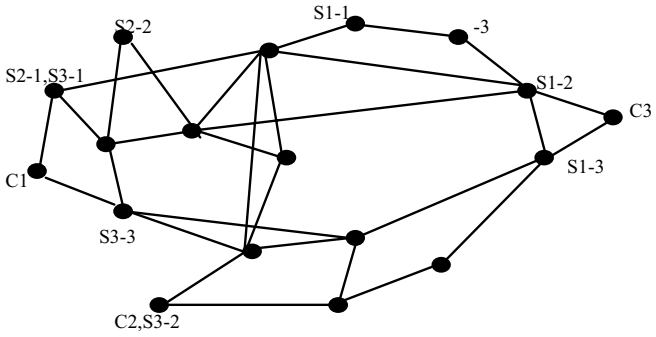


Fig. 4. Topology for nonuniform traffic. The three hot pairs are: (1) $C1 : \{S1-1, S1-2, S1-3\}$, (2) $C2 : \{S2-1, S2-2, S2-3\}$, and (3) $C3 : \{S3-1, S3-2, S3-3\}$. Assume that clients and servers are directly behind the router nodes being addressed

5. Performance evaluation

The objective of our performance study is twofold: (i) to study in detail the performance of policy-based CPSS algorithms under various request patterns and load situations and (ii) to correlate the performance results to the information collection costs under various directory service update techniques. This will help us understand the dynamics and the tradeoffs underlying our CPSS algorithm in a distributed environment.

5.1. The simulation model and environment

We use a simulator call "QSim", which was developed at UC Irvine. QSim is a message-driven multithreaded simulator intended to study the dynamics of QoS-sensitive network environments. QSim has the following components: traffic source, routers, servers, and directory service nodes. The reservation in QSim is rather simple; the actual implementation could use standard RSVP to make reservation in a distributed global network. Because QSim does not go into the details of packet passing, we model the delay characteristic of a link and a server as exponentially correlated to their residue capacities.

Topology and system configuration. In the simulation, we use a typical ISP network topology with 18 nodes and 30 links, as illustrated in Fig. 4. We assume that each node is a network router and that the clients and servers are distributed throughout the network and are directly behind the router nodes (not shown in the graph). The topology is chosen such that there are a large number of alternative paths between source and destinations nodes.

To better emulate the real network, the capacities of network links are selected from various widely used link types from 1.5 Mbps to 155 Mbps, with the mean value being 64 Mbps. When defining the capacity of the server nodes, we calibrate CPU units using a basic 64-kbit voice processing application, memory units to be 64 kbytes, and disk bandwidth units to be 8 kbytes/s. The server capacities are also selected from popular models of multimedia servers and Web servers, with the CPU, memory, and disk bandwidth mean to be 1845, 6871, and 5770 calibrated units, respectively.

Request and traffic generation model. We model request arrival at the source nodes as a Poisson distribution,[4] and the request holding time[5] is exponentially distributed with a prespecified average value. We predefine a set of client request templates to capture typical multimedia connection request patterns in terms of network bandwidth, CPU, memory, disk bandwidth, and end-to-end delay. For each request generated, the requested parameters are randomly selected from the set of request templates, with the mean requested bandwidth being 2.5 Mbps, mean end-to-end-delay 400 ms, and CPU, memory, and disk bandwidth 150, 374, and 271 calibrated units, respectively. To model traffic, we generate two types of traffic patterns: nonuniform traffic and uniform traffic. To represent nonuniform traffic, we designate some sets of candidate destinations as being "hot" (i.e., serving popular videos, Web sites, etc.); such destinations are selected by clients more frequently than others. To reduce the effect of local and nearby requests, we choose three pairs of source–destination sets from the topology. The requests arrive at these hot pairs as foreground traffic at a higher rate than other background traffic. In our nonuniform traffic pattern, we set the foreground arrival rate to be five times higher than the background rate, and in uniform traffic patterns we set them equal to one another. Specifically, we set the foreground arrival rate to 10 s and the background rate to 50 s. To simulate medium-sized multimedia sessions, we set the average request hold time to 10 min.

5.2. Path and server scheduling

Given a set of feasible assignments, we start by studying the following policies for choosing the optimal assignment: X^*, best UF, shortest-hop, random, Prob-1ϕ, and Prob-2ϕ.

Best UF policy. The performance of the best UF policy under various information update frequencies is depicted in Fig. 5a. Given near current system state information, the best UF policy is a variation of online algorithms that optimize the current assignment (i.e., maximize overall resource utilization) without knowledge of future requests. With near current system state information, i.e., with very short update periods, we notice that requests are rejected by the directory service (DS) when it tries to find an assignment for the requests and encounters limitations in the network and server capacity. Most requests admitted by the DS are eventually committed by network and servers, so the network and server rejection rate is very low, resulting in a higher utilization of the network and server.

When the state information in the DS is very inaccurate, i.e., long update period, the directory uses the outdated load information to assign the *same* "best" paths and servers to the clients, which quickly congests these nodes and links. This causes the network or server to reject the assignment chosen by the DS. Request rejections at the server and network cause the overall system utilization to fall; the drop in utilization is recorded in the DS during an update. We also noticed that server rejection appears to dominate the overall performance when the DS is inaccurate since server resources are fewer than network resources and are saturated more quickly (Fig. 5b).

Shortest-hop policy. This policy is interesting because it provides a shortest widest solution, i.e., the policy chooses the nearest feasible server from the client in terms of number of hops. If multiple such servers exist, the policy chooses the least loaded one, i.e., the one with the lowest UF value. In general, we can see that the performance of the shortest-hop policy is worse than best UF (see comparative performance in Fig. 8). This is because best UF subsumes shorter, wider paths. The longer the path, the bigger the resulting UF value since for a path p, $UF(p) = \sum UF(l), l \in p$. The shortest-hop policy only considers the length of the path and thus tries to optimize the usage of network resources by using the least number of network links without considering current load situations on the links or servers. When state information is not current, the shortest-hop policy tends to initiate congestion earlier than best UF because it potentially overloads some paths and servers that are already very congested. Hence a large number of request rejections in the shortest-hop policy result from path rejections in the network (Fig. 6a).

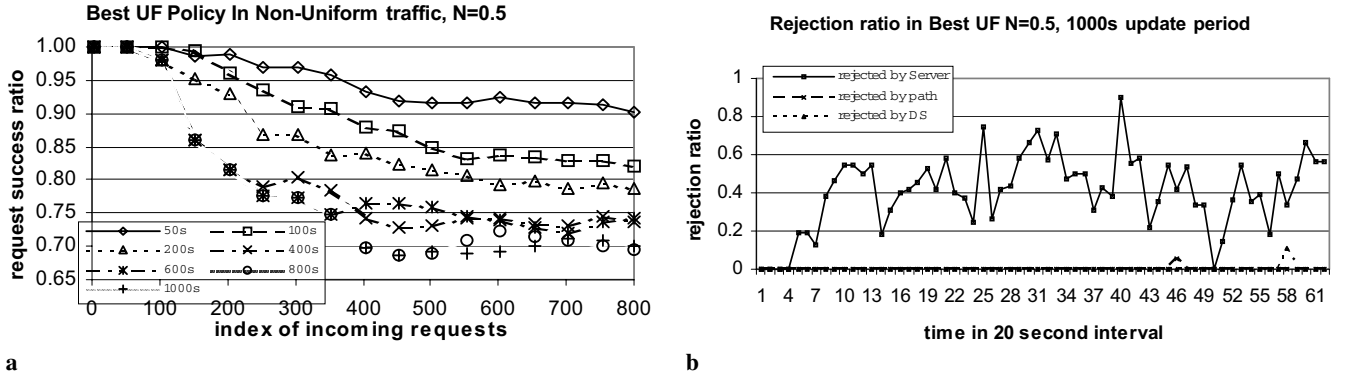
Random policy. This policy tries to avoid the oscillation of static policies and balances the load of the system by randomly picking one choice from the feasible assignment set X calculated by CPSS. While the random policy balances the load between servers, it does not use the load information of the assignments and hence often results in longer network paths, with the request getting quickly rejected by the network nodes. Figure 6b shows that the path rejection dominates the overall request/reject ratio. We notice that the random policy performs consistently worse than the best UF and shortest-hop policies because it does not differentiate between all the feasible assignments in the set X and repeatedly picks some feasible but marginal assignment, leading to congestion.

The weighted differential probabilistic policy (Prob-1ϕ). Here, an assignment with a lightly loaded server has a high probability of being selected even if the network path leading to that server is highly congested. This causes a large number of network path rejects (Fig. 6c). This side effect is eliminated in the second phase of the Prob-2ϕ policy where all paths leading to that server are weighted according to their residue capacity (see Fig. 8 for overall performance).

The load-sensitive differential probabilistic policy (Prob-2ϕ). Prob-2ϕ differentiates between the network and server resources and ensures that the more bottlenecked resource is selected with a lower probability. We further experimented with variations of the Prob-2ϕ policy using either a weighted or uniform (i.e., all servers weighted equally) probabilistic method when choosing an optimal assignment from the feasible server set. Uniform and weighted probabilistic policies produce varying effects under a large update timer. In the presence of congestion, the weighted policies tend to provide better load balance in the short term

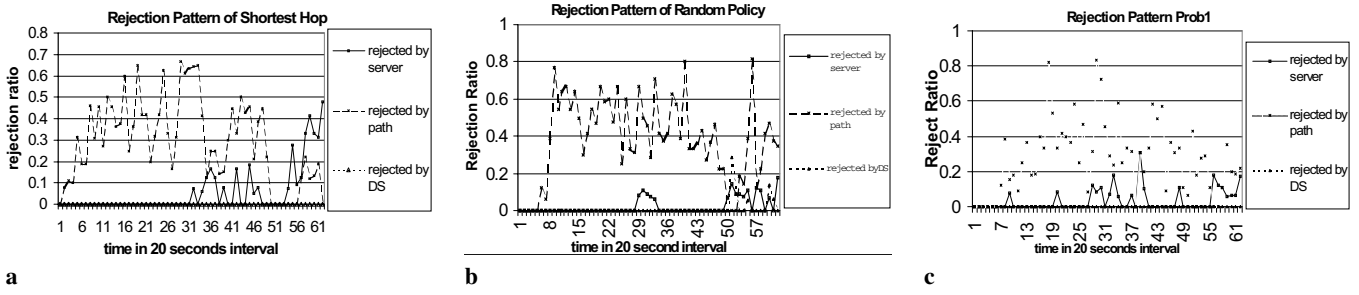
[4] Since our study focuses on generalized traffic patterns, we do not model requests for the specific MM objects. We intend to explore more sophisticated traffic generation models (e.g., Zipf-like) that account for popularity-based generation of specific requests when we consider object placement techniques and scheduling specific requests for objects.

[5] The request holding time is the time for which the requested network and server resources such as link bandwidth, CPU, buffer, disk bandwidth, etc. are reserved.



a b

Fig. 5a,b. Best UF policy in nonuniform traffic environments. **a** With different update periods. **b** Rejection pattern with update period $= 1000$ s



a b c

Fig. 6a–c. Rejection pattern for CPSS policies with very infrequent sampling (once per 100 s). **a** Shortest-hop policy. **b** Random policy. **c** Prob-1ϕ policy

(Fig. 7b); however, they can cause further congestion over a period of time. The uniform policy will not alter existing load conditions dramatically, resulting in more rejects in the short term (Fig. 7a). In our experiments, we noticed that the server and network rejects alternate and are comparable in number.

5.3. Information update

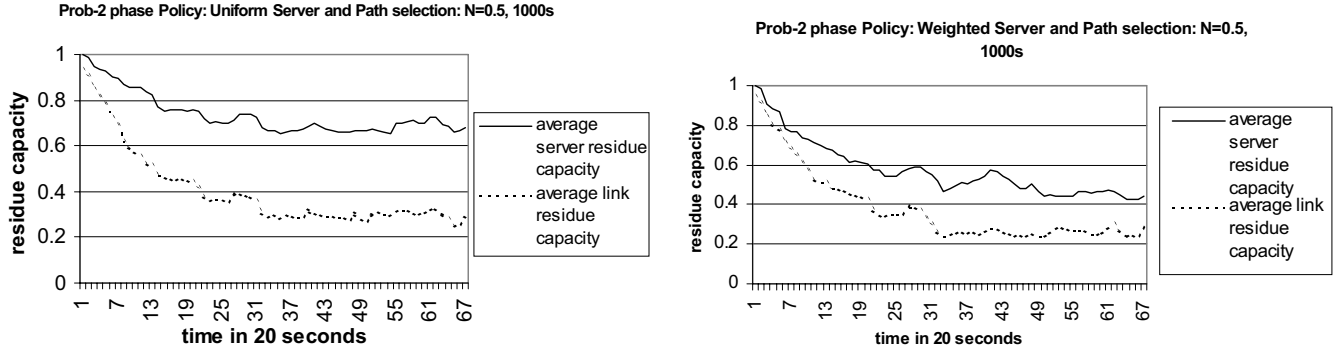
To further evaluate the performance and efficiency of our previous CPSS policies, we compare them with a static non-CPSS "nearest" server algorithm, which implements server selection by counting the number of hops from the client to all candidate servers and selecting the nearest one. The results in Fig. 8 show that, in nonuniform traffic environments, the CPSS policies are 20%–30% better than the static algorithm, while in a uniform traffic environment, CPSS policies outperform the static algorithm by 15% on average. The performance gain of the CPSS algorithm is obtained at the cost of increased message overhead caused by periodic system state updates. The update period dominates the information accuracy in the directory and thus influences the performance of the CPSS algorithm. In the following section we focus on the influence of the information update overhead on CPSS policies.

Snapshot-based information update. The snapshot-based information update can be regarded as a special case of the interval-based update with the range to 1. In general, we observe that a shorter update period results in better performance than a larger update period. This holds only when the information update period is smaller than the average connection holding time. For instance, in our experiments the average connection holding time is set to 10 min (600 s). Update period values larger than 600 s show negligible effects on the performance of CPSS policies.

If the system state information is updated very frequently, our experiments indicate that the best UF policy is superior. This is because the best UF policy tries to find an optimal tradeoff between the shortest and widest paths. In situations where information update is infrequent, the deterministic policies, best UF and shortest-hop, inevitably go into oscillation, limiting the overall system throughput. The nondeterministic policies – random, Prob-1ϕ, and Prob-2ϕ – distribute load among multiple feasible servers and network paths, thus preventing the oscillation and improving the overall resource utilization. It should be noted that there are significant performance differences among these three policies; Prob-2ϕ consistently performs best and the random policy always performs worst.

Interval-based information update. To further save the cost of information update, we introduced interval-based information update policies; the reduction in message overhead cost can be seen in Fig. 3. Furthermore, we believe that it is natural to represent a residual value using ranges, especially when infrequent updates are desired. Our simulation shows that the interval-based policies do not have a significant influence when used together with deterministic CPSS policies. Hence we restrict our description to the study of interval-based update policies when used together with probabilistic policies, i.e., Prob-2ϕ.

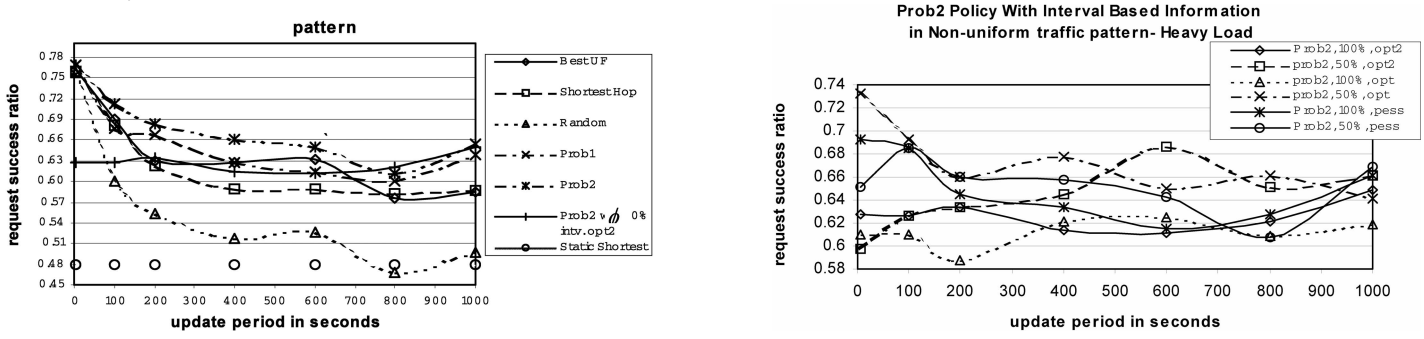
Figures 8b,d,f show the performance of interval-based update policies in various traffic environments. With frequent updates, a smaller range for the interval (50%) of the maximum request size) brings better CPSS performance than a bigger



a **b**

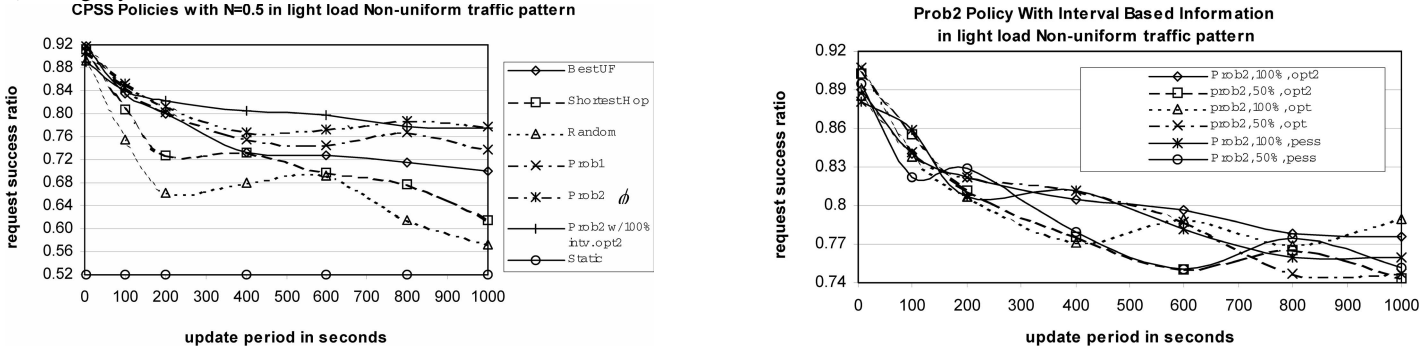
Fig. 7a,b. The Prob-2ϕ policy in nonuniform traffic environments, update period $= 1000$ s. **a** Servers weighted equally (*left*). **b** Servers weighted according to their UF values (*right*)

a,b In heavily loaded nonuniform traffic environment



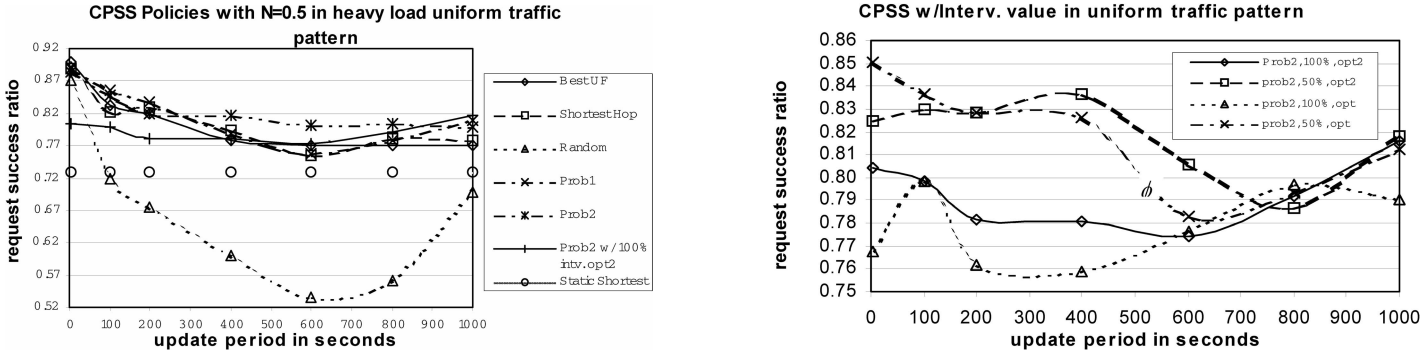
a Overall performance. **b** Interval-based update policies with Prob-2ϕ

c,d In lightly loaded nonuniform traffic environment



c Overall performance results. **d** Interval-based update policies with Prob-2ϕ

e–f In uniform traffic pattern



e Overall performance results. **f** Interval-based update policies with Prob-2ϕ

Fig. 8a–f. Overall performance of CPSS with information update

interval (100% of the maximum request size), as illustrated in Figs. 8b and f. For a larger update period, a bigger range brings better CPSS performance. The reason is that when the update period is short, representing a residue value that uses a big range introduces information inaccuracy, and a shorter range is better. However, when the update period is very long, a residue value represented by a small range often gets outdated sooner than a larger one, resulting in a more inaccurate system state information. In general, under both lightly and heavily loaded conditions, we notice that the performance of Prob-2ϕ is better than other CPSS policies with either interval-based or snapshot-based updates. Specifically, we show in Figs. 8a, c and e that Prob-2ϕ with 100% interval achieves comparable performance with Prob-2ϕ under snapshot-based update in large update timers. In a lightly loaded nonuniform traffic environment, the variations of interval-based update do not exhibit an obvious influence on the performance of Prob-2ϕ (Fig. 8d). Of the interval-based updating techniques (PESS, OPT, and OPT2), from Figs. 8b and f we show that OPT2 performs best in most cases. In a lightly loaded environment as shown in Fig. 9d, the difference is not obvious because the bandwidth and server resources are largely sufficient relative to user requests.

In summary, our evaluation indicates that CPSS-based policies perform about 20%–30% better on average than non-CPSS policies where server selection is performed using a static, nearest-server policy. This holds true in both nonuniform and uniform traffic environments. With a high update frequency and up-to-date state information, best UF, shortest-hop, and Prob-2ϕ policies perform comparably; with medium to low update frequencies, Prob-2ϕ performs consistently better. In addition, our study indicates that when information is updated less often, interval-based updating mechanisms are more cost effective than snapshot-based techniques since they can achieve comparable performance with lower message overheads.

6. Related work and conclusions

QoS-based resource allocation has been addressed independently within the networking, multimedia, and distributed computing communities. We focus on two areas – replicated service selection and QoS routing. Static nearest-server algorithms [22] attempt to choose a replica server with the minimum number of hops from the client, with topology information being stored in a topology database. Such policies do not consider server quality as a scheduling parameter, which results in performance degradation in some situations. On the other hand, dynamic server selection policies have been studied in the context of replicated Web services [15,30,7,14] and load balancing for multimedia streaming applications [35,24]. While server selection and load balancing mechanisms address effective utilization of server resources, the CPSS approach presented in this paper also accounts for network factors that play a role in delivering end-to-end application QoS. It therefore prevents the situation where a poor network connection can deteriorate service from the optimal server.

QoS-based routing techniques have been explored in depth [9,10,5,3]. QoS-based extensions of the traditional Bellman-Ford/Dijkstra algorithm [4] incorporate an available bandwidth parameter and use widest shortest path [20,8] calculations. Experimental efforts studying the combined performance and overhead cost of different information collecting and routing policies [1] show that maintaining a large number of alternate paths and randomizing the path selection will yield better performance in most topologies and traffic patterns, especially when information update is very infrequent. Link parameters such as delay, available bandwidth, etc. can be described using probability distributions [29] and can be used to find a *most probable* path satisfying the requested bandwidth and end-to-end delay. A heuristic to the most probable-optimal partition problem has a complexity of $O(|E|^2 D)$, which uses a dynamic programming solution developed in the context of RSP problems [23].

Our work differs from traditional QoS routing in several respects. First, we present and evaluate a directory-enabled approach to provide combined path and server selection in dynamic multimedia environments. We exploit architectural considerations of network links and servers to treat them differentially in the overall path/server selection process. Through extensive studies, we have determined suitable CPSS policies that can adapt to dynamic network and system conditions. Specifically, we have studied parameter update mechanisms such as range-based update policies in the directory service for effective performance of the CPSS strategies. More recently, content delivery infrastructures (CDIs) have been proposed within the IETF to support an overlay network infrastructure beyond the current Internet for smooth delivery of content (e.g., images, streaming video, Web pages) to a large client population. This paper assumes an underlying framework similar to the CDI and develops a unified resource provisioning algorithm that integrates the considerations of network and server load to improve the overall systemwide performance.

In this paper we have demonstrated the effectiveness of an integrated approach to QoS-based resource provisioning in a distributed environment. While prior work has categorized functionality that must be addressed at different levels in the system (e.g., network layer routing, server-based load balancing), our approach provides a more holistic approach to managing emerging distributed applications where end-to-end service guarantees must be enforced at multiple levels. The proposed middleware solution uses a directory service component that maintains reasonably accurate information of the network and server system state; this solution ensures that composite path and server provisioning is performed. Intelligent CPSS policies and effective DS maintenance strategies ensure good performance despite statistical fluctuations in network and system conditions.

Much work remains to be done in ensuring the scalability and manageability of a directory-based framework. We are exploring variations of the range-based update techniques such as exponential interval-based and adaptive dynamic range-based techniques [18]. While the earlier techniques deal with fixed ranges and are history insensitive, the adaptive dynamic range-based scheme alters the range dynamically using a history-sensitive update procedure [18,19]. In addition, in this paper we assume a given replica/placement model that is used in server and path selection. We intend to explore specific placement policies using dynamic replication and migration in a wide-area scenario. We hope to eventually apply and evaluate the developed techniques to provide seamless delivery

of multimedia content in mobile and wireless environments [36,37,32]. Further work on QoS-based provisioning in wide-area distributed environments is being studied in the context of the AutoSeC project. AutoSeC (Automatic Service Composition) [26,25] is an integrated middleware framework in highly dynamic environments, e.g., mobile and sensor-based networks where applications exhibit real-time and security requirements. We believe that the cooperative execution of multiple management mechanisms is the key to effective system utilization. The work presented in this paper is a step toward trying to integrate such policies into a uniform framework.

Acknowledgements. The authors would like to thank the members of the DSM group at UC Irvine, especially Qi Han for useful comments and suggestions. This work was supported by funding from a Career Award (NSF ANI-9875988).

References

1. Apostolopoulos G, Guerin R, Kamat S, Tripathi SK (1998) Quality of service routing: a performance perspective. In Proceedings of ACM Sigcom'98, location, day month 1998, pages
2. Barbir A, Cain B, Douglis F, Green M, Hoffmann M, Nair R, Potter D, Spatscheck O (year) Known CDN request-routing mechanisms. draft-ietf-cdi-known-request-routing-00.txt (WiP)
3. Breslau L, Estrin D, Zhang L (1993) A simulation study of adaptive source routing in integrated service network. USC CSD Technical Report Sep
4. Bertsekas D, Gallager R (1992) Data networks. Prentice-Hall, Englewood Cliffs, NJ
5. Breslau L, Shenker S (1998) Best-effort versus reservations: a simple comparative analysis. In: Proceedings of ACM Sigcomm '98 location, day month 1998, pages
6. Braden R, Zhang L, Berson S, Herzof S, Jamin S (1997) Resource reservation setup protocol (RSVP): version 1: Functional specifications. RFC 2205
7. Carter RL, Crovella ME (1997) Dynamic server selection using bandwidth probing in wide-area networks. In: Proceedings of INFOCOM '97, location, day month 1997, pages
8. Chen S, Nahrstedt K (1998) On finding multi-constrained paths. In: Proceedings of IEEE international conference on communications (ICC 98), Atlanta, GA, day month 1998, pages
9. Chen S, Nahrstedt K (1999) Distributed quality-of-service routing in ad-hoc networks. IEEE J Special Areas Commun Special Issue on Ad-Hoc Networks volume:pages
10. Cidon I, Rom R, Shavitt Y (1997) Multi-path routing combined with resource reservation. In: Proceedings of INFOCOM'97, location, day month 1997, pages
11. Day M, Cain B, Tomlinson G, Rzewski P (2002) A model for content internetworking (CDI). Internet draft draft-ietf-cdi-model-02.txt (WiP), 3 May 2002
12. Dan A, Dias D, Mukherjee R, Tewari R (1955) Buffering and caching in large scale video servers. IEEE COMPCON
13. Day M, Gilletti D, Rzewski P (2002) Content internetworking scenarios. Draft-ietf-cdi-scenarios-00.txt (work in progress), February 2002
14. Fei Z, Bhattacharjee S, Zegura EW, Ammar MH (1997) A novel server selection technique for improving the response time of a replicated service. In: Proceedings of INFOCOM'97, location, day month 1997, pages
15. Francis P, Jamin S, Paxson V, Zhang L, Gryniewicz D, Jin Y (1999) An architecture for a global internet host distance estimation service. In: Proceedings of INFOCOM'99, location, March 1999, pages
16. Fei A, Pei G, Liu R, Zhang L (1998) Measurements on delay and hop-count of the internet. In: Proceedings of GLOBECOMM '98, location, day month 1998, pages
17. Fu Z, Venkatasubramanian N (1999) Combined path and server selection in dynamic multimedia environments. In: Proceedings of ACM MM '99, location, day month 1999, pages
18. Fu Z, Venkatasubramanian N (2001) Directory based information collection for QoS provisioning in dynamic multimedia environments. In: Proceedings of the IEEE international conference on distributed computing systems (ICDCS 2001), location, day month 2001, pages
19. Fu Z, Venkatasubramanian N (2001) An evaluation of composite routing and scheduling policies for dynamic multimedia environments. In: Proceedings of the IEEE international conference on parallel and distributed processing systems (IPDPS 2001), location, day month 2001, pages
20. Roch A, Guerin AO, Douglas W (1998) QoS routing mechanisms and OSPF extensions. In: Proceedings of BLOBCOM'98, location, day month 1998, pages
21. GTE Internet Access Services (1999) Managed hosting with traffic distributor. http://www.bbn.com/services/hosting/traffic
22. Guyton JD, Schwartz MF (year) Locating nearby copies of replicated internet servers. In: Proceedings of ACM SIGCOMM, location, day month year, pages
23. Hassin R (1992) Approximation schemes for the restricted shortest path problem. Math Op Res volume:pages
24. Harchol-Balter M, Crovella ME, Murta CD (1998) On Choosing a task assignment policy for a distributed server system. In: Proceedings of Performance Tools '98, location, day month 1998. Lecture notes in computer science vol 1468. Springer, Berlin Heidelberg New York, pages
25. Han Q, Venkatasubramanian N (2001) AutoSeC: An integrated middleware framework for dynamic service brokering. IEEE Distr Sys Online 2001 2(7)
26. Han Q, Venkatasubramanian N (2002) A cost driven approach to information collection for mobile multimedia environments. In: Proceedings of the IEEE international conference on mobile and wireless communications networks (MWCN 2002), location, day month 2002, pages
27. Keshav S, Sharma R, Siamwalla R (1998) Project octopus: network topology discovery. http://www.cs.cornell.edu/cnrg/topology/Default.html
28. Lang K, Rao S (1993) Finding near-optimal cuts: an empirical evaluation. In: Proceedings of the 4th annual ACM-SIAM symposium on discrete algorithms, Austin, TX, January 1993, pp 212–221
29. Lorenz DH, Orda A (1998) QoS routing in networks with uncertain parameters. In: Proceedings of INFOCOM 1998, location, day month 1998, pages
30. Myers A, Dinda P, Zhang H (1999) Performance characteristics of mirror servers on the Internet. In: Proceedings of GLOBECOM 1999, location, day month 1999, pages
31. Ma Q, Steenkiste P, Zhang H (1996) Routing high-bandwidth traffic in max-min fair share networks. In: Proceedings of SIGCOMM 1996, location, day month 1996, pages
32. Mohapatra S, Venkatasubramanian N (2003) PARM: Power-aware reconfigurable middleware. In: Proceedings of the IEEE international conference on distributed computer systems (ICDCS 2003), location, day month 2003, pages

33. Venkatasubramanian N, Deshpande M, Mohapatra S, Gutierrez-Nolasco S, Wickramasuriya J (2001) Design and implementation of a composable reflective middleware framework. In: Proceedings of the IEEE international conference on distributed computer systems (ICDCS-21), location, day month 2001, pages
34. Rosen E, Viswanathan A, Callon R (2001) MPLS architecture. RFC3031
35. Venkatasubramanian N, Ramanathan S (1997) Load management for distributed video servers. In: Proceedings of the international conference on distributed computing systems (ICDCS'97),location, day month 1997, pages
36. Wolfson O, Chamberlain S, Dao S, Jiang L, Mendez G (1998) Cost and imprecision in modeling the position of moving objects. In: Proceedings of the international conference on data engineering, location, day month 1998, pages
37. Wolfson O, Sistla P, Dao S, Narayanan K, Raj, R (1995) View maintenance in mobile computing. SIGMOD REC volume:pages
38. Zhao W, Tripathi SK (1998) Routing guaranteed quality of service connections in integrated service packet network

Appendix I – Feasibility proof of CPSS

Lemma 1. If path $P_n, P_n = \{(O, u_{n,1}), (u_{n,1}, u_{n,2}), \ldots, (u_{n,k-1}, u_{n,k}), (u_{n,k}, CD)\}$ is the RSP from origin O to common destination (CD) with the delay n. There will be one and only one server node s on path P_n and it must be $u_{n,k}$.

Proof. When constructing graph G', we have $\forall(u, CD) \in E'$, and u is a server node. This shows that there is at least one server node on P_n, and on the path $\{(O, u_{n,1}), (u_{n,1}, u_{n,2}), \ldots, (u_{n,k-1}, u_{n,k}), (u_{n,k}, CD)\}$, node $u_{n,k}$ is a server node. Suppose there are two servers s and s' on path P_n. This implies that there is an edge from a server node s' to some vertices $u_{n,j}$, $(s', u_{n,j})$. But the server nodes do not have outgoing edges when constructing the graph G', and this is impossible. This proves the Lemma. □

Theorem 1 An assignment with end-to-end delay d, $X_i^d :< p_i, s_i >$, where $p_i = \{(O, u_1), (u_1, u_2), \ldots, (u_{k-1}, s_i)\}$, satisfies the feasibility condition if $\mathrm{DIST}_{CD}[d] < \infty$, $d \leq DL_r$, and $P_d = p_i \cup (s_i, CD)$, where P_d is the corresponding feasible path with delay d.

Proof. If $\mathrm{DIST}_{CD}[d] < \infty$, and P_d is the corresponding path, $P_d = p_i \cup (s_i, CD)$. We show that an assignment $X_i^d :< p_i, s_i >$ derived from P_d satisfies the feasibility condition. (1) Feasibility conditions 1 and 2 are satisfied; otherwise the links and server nodes would have been removed from graph G', and because P_d is a path of graph G', so p_i and s_i satisfy the first two feasibility conditions. (2) Feasibility condition 3 is satisfied because $d < DL_r$, $\mathrm{DIST}_{CD}[d] < \infty$ and $Delay(P_d) = EED^{X_i^d} = DL^{p_i} + RSP^{s_i} \leq DL_R$. □

Theorem 2 The CPSS algorithm finds an optimal assignment in $O(|DL_r|E')$

Proof. From a dynamic programming table structure, the update is done for each outgoing edge of a vertex for each delay constraint value. Thus the total time complex is $O(|DL_r|E')$.