An Evaluation of Composite Routing and Scheduling Policies for Dynamic Multimedia Environments

Zhenghua Fu and Nalini Venkatasubramanian
Department of Information and Computer Science
University of California, Irvine , Irvine CA 92697-3425
{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 combined path and server selection (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 number of requests. Multimedia applications require Quality of Service (QoS) guarantees; resource provisioning techniques for multimedia applications must ensure that Quality-of-Service 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. Quality of Service (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 pre-determined 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 the 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 system-wide 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, using which combined path and server selection (CPSS) problem can be tackled effectively. Specifically, we

(a) develop a model and algorithm to address the combined path and server selection (CPSS) problem based on system residue capacities [FV99].

(b) develop and evaluate a family of directory-enabled policies for composite routing and scheduling [FV01-2].

(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 case of the CPSS algorithm. In Section 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 Section 7 concludes with future research directions.

2. COMBINED PATH AND SERVER SELECTION (CPSS)

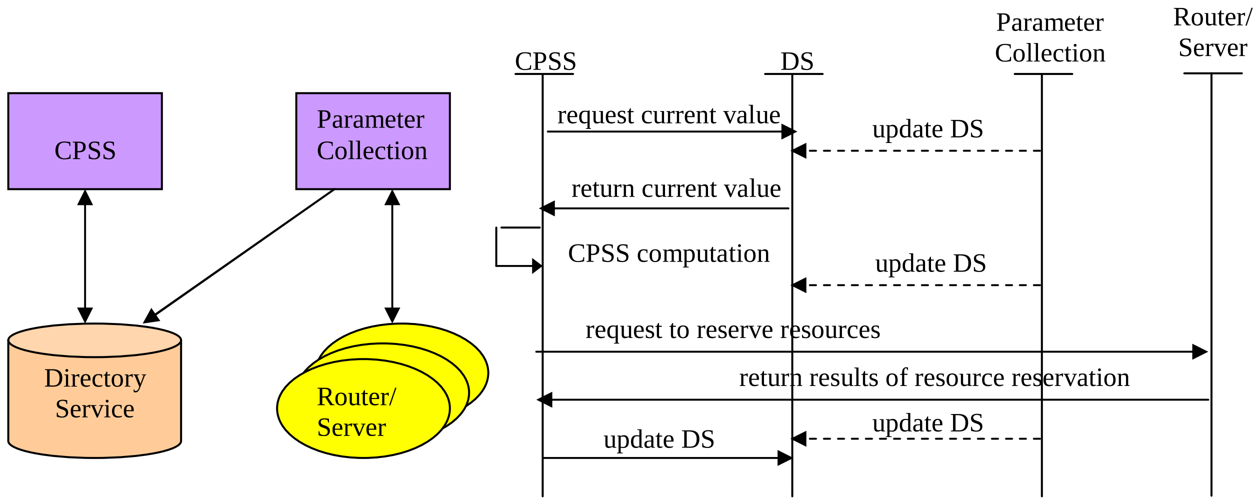


Fig 1: *System Architecture and Operational Flow of the CPSS Process*

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 which 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 (described in Section 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:$< PATH_R, SERV_R, ETOE_R >$.

The path QoS requirement is defined to be 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,

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

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 (See Figure 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 note 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

$$BW^p_{avail} = \underset{l \in p}{Min}\left\{BW^l_{avail}\right\};\ DL^p = \sum_{l\in p} DL^l .$$

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.

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

In order 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

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

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

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

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 [LR93]. It serves as a tuning knob to bias resource allocation in favor of lightly loaded resources and is discussed in more detail in the appendix.

For an assignment *X={p,s}*, we define the distance of the server *s* to be

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

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 set of all the assignments that meet the feasibility condition.

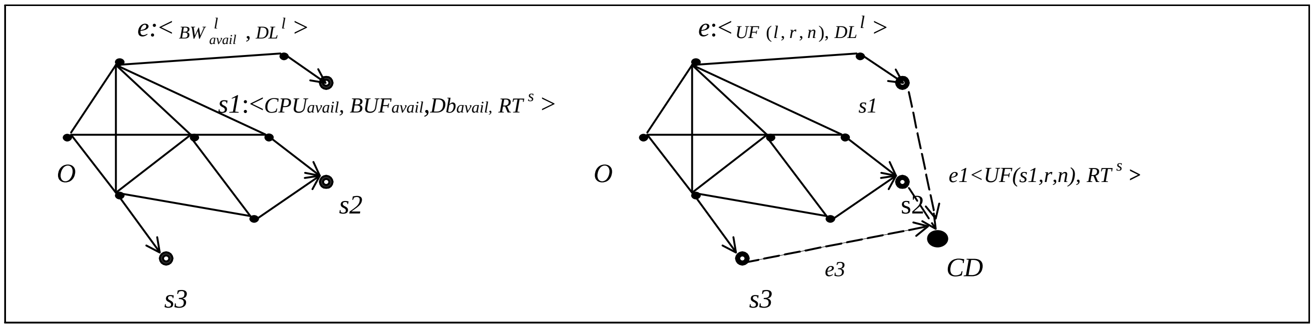

$e:< BW^l_{avail}, DL^l >$
$s1:< CPU_{avail}, BUF_{avail}, Db_{avail}, RT^s >$
O
s2
s3
$e:< UF(l,r,n), DL^l >$
s1
O
$e1< UF(s1,r,n), RT^s >$
s2
e3
CD
s3

***Figure 2. CPSS Diagram**. Left: Graph G:<N,E> with the client requesting at point O and a set of target servers S: s1,s2,s3. Right: Graph extended from G, adding a point CD and artificial edges e1,e2 and e3*

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 criteria. For instance, the optimality clause for the BEST UF policy is

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

We will discuss specific CPSS policies in detail later in Section 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 Figure 1 above].

The weight of *e* is defined as *W(e):<UF,DL>*, 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*.

```
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 Restricted Shortest Path 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*\(s*, CD), s*}, based on CPSS policy
4. Return X*
```

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 [H92], there exist heuristic techniques, (e.g. dynamic programming), to solve it by assuming an integer value of the delay constraint [H92, LO98, CN98]. 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 represent the maximum utility factor from source to a certain node (the corresponding column) within a delay constraint (the 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 [LO98]. 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 Restricted Shortest Path (RSP) algorithm is $O(D|N|)$.

3 A FAMILY OF CPSS POLICIES

We propose deterministic and non-deterministic 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 s.t. the number of hops from source to destination is minimal: *X*:<p*,s*> s.t. Hop(X*) = Min{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*> s.t.* $Dist(X^*) =$ *UF(p*)+UF(s*)=Min(Dist(X) | for all X 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.

Non-deterministic CPSS Policies: Non-deterministic policies attempt to achieve a better degree of load balance among the links and servers. We present 3 non-deterministic 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-1 ϕ): 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

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

R1 and *R2* decide how the server and path resources should be emphasized in calculating the selection probability of the assignment. For instance, in order 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-1 ϕ (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,…,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-2 ϕ): 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 constitute the bottleneck for the client's request. 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 on 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 value among all such paths.

(b) If the network is the restricting element, we emphasize on 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 hotspots and maximize load balance among the various network paths.

```
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 more heavily loaded than the network */
        Weight server $s_i$ in S as: $\frac{UF^{-1}(s_i)}{\sum_{j\le k} UF^{-1}(s_j)}$
        Select a server s* from S with max weight

4. /* Select a best path leading to the selected server */
     Determine set $X' \subseteq X$, s.t. $X'=\{X_i:<p_i,s_i> \mid s_i = s^*, i=1,2,\ldots n.\}$
     Weight $X_i$ in X' as $\frac{UF^{-1}(p_i)}{\sum_{j\le k} UF^{-1}(p_j)}$
     Select an assignment X* from X' probabilistically
5. Return the selected assignment X*
```

[1] Here, in order to be comparable to server, $UF(p_i) = Max\{UF(l) \mid l \in p_i\}$.

4 DIRECTORY ENABLED PARAMETER COLLECTION AND UPDATE

The CPSS policies discussed are based on the knowledge of network topology, replica maps, and the 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];
- 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 the 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 and (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 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 framework proposed 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 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 [AGKT98, FV99, FV01-1]; 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>=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 the DS range, the directory is updated with another interval based range, otherwise no update is sent.

[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 ([FJPZGJ99], [KSS98]).

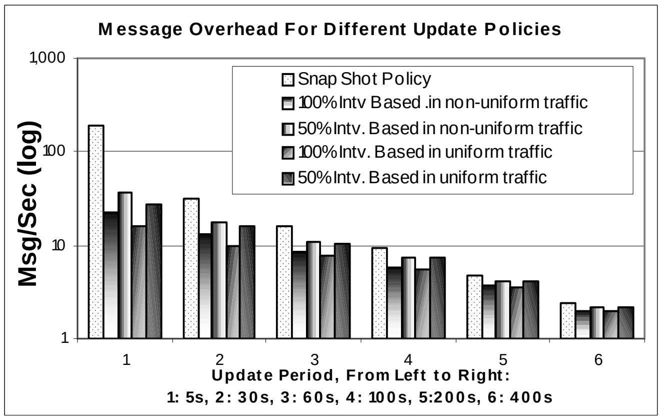


Fig 3: Message Per Seconds For Update Policies

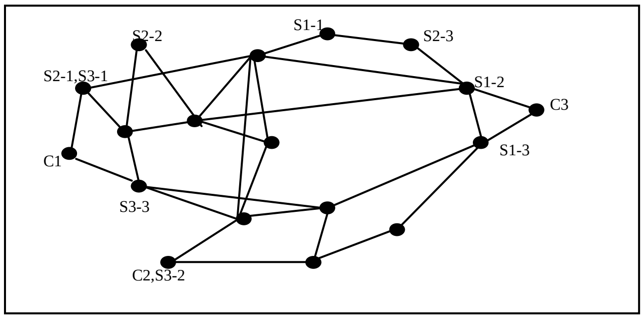


Fig 4. Topology - For *non-uniform traffic*, the three hot pairs are: 1) C1:{S1-1,S1-2,S1-3}, 2) C2: {S2-1,S2-2,S2-3}, 3) C3: {S3-1,S3-2,S3-3} Assume that the clients and servers are directly behind the router nodes being addressed.

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 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 second period), the message overhead cost for the instantaneous snapshot-based representation is about 10 times that with an interval based representation. When update frequency is very low, (e.g., with a 400 seconds 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 can not be used in CPSS calculations, we need to convert a range, say [*L*, *H*], to a number. We further explore 3 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 (in the following 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 directory service so that the requested resource is pre-deducted along the assigned path/server from the state information database prior to the actual resource reservation process. There are a several disadvantages 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 and range-based approaches to independently maintain the DS. In particular, we use a static interval-based collection processes that use fixed size ranges and sampling frequencies in the following performance evaluation section. In [FV01-1], we explore more dynamic information collection techniques with variable frequency and adjustable range representation using an auto-regressive 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 have developed a detailed analysis on how the specific statistical distributions (e.g. Poisson or Pareto) affects the accuracies and overhead cost of our range based information collection technique (the interested reader is referred to [FV01-1]). In the following, we provide a brief description on how both small and dramatic statistical fluctuations are accommodated by the proposed framework.

Small fluctuations in system state: In case 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 the information collection and place focus on the discussion of the design and evaluation of CPSS policies. In other work[FV01-1], 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 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 which explicitly tries to eliminate hot spots among networks and replica servers and distribute the overall load evenly among 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 places 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.

5 PERFORMANCE EVALUATION

The objective of the simulation is to study in detail the performance of policy based CPSS algorithms under different request patterns and load situations and to correlate the performance results to the information collection costs. This will help us understand the dynamics and the tradeoffs underlying our CPSS algorithm in a distributed environment. Our performance study consists of 2 steps:

(a) Evaluation of policies for optimal path and server selection in the CPSS process: We focus on different policies for *optimal* path and server assignment among multiple feasible assignments which all satisfy our base CPSS requirements. In the previous discussion, the base CPSS algorithm finds all feasible assignments that satisfy the QoS requirements of the client, which includes network and server parameters as well as end to end delay requirements.

(b) Evaluation of directory service update techniques for CPSS decision making: Our simulation will further focus on situations where system state information is updated very infrequently, i.e., the information collection period is relatively long, to study the cost-effectiveness of different polices in different traffic patterns and load situations.

5.1 The Simulation Model and Environment

We use a simulator call "QSim", developed at UC Irvine. QSim is a message driven multi-threaded simulator intended to study the dynamics of QoS sensitive network environments. Since we are interested in the overall behavior of traffic flows, in particular, flow setup and termination, the simulator doesn't deal with details of packet transmission in the network. 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 doesn't 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 (See Appendix on Deriving Delay Values).

Topology and System Configuration: In the simulation, we use a typical ISP network topology with 18 nodes and 30 links as illustrated in Figure 4. We assume that each node is a network router, and that the clients and servers are distributed among 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 1.5Mbps to 155Mbps, with the mean value being 64M. When defining the capacity of the server nodes, we calibrate CPU units using a basic 64Kbit voice processing application, memory units to be 64Kbytes, and disk bandwidth units to be 8Kbytes/s. The server capacities are also selected from popular models of multimedia servers and web servers, with the CPU, memory, 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 pre-specified average value. We pre-define a set

[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.

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 2.5Mbps, mean end-to-end-delay being 400ms and CPU, memory and disk bandwidth being 150, 374 and 271 calibrated units respectively. To model traffic, we generate two types of traffic patterns: non-uniform traffic and uniform traffic. To represent non-uniform traffic, we designate some sets of candidate destinations as being "hot", (i.e. serving popular videos, web sites etc), and they are selected by the 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 to these hot pairs, as foreground traffic, at a higher rate than other background traffic. In our non-uniform traffic pattern, we set the foreground arrival rate to be 5 times higher than the background rate, and in uniform traffic pattern, we set them equal. Specifically we set the foreground arrival rate to be 10 seconds, and the background rate to 50 seconds. In order to simulate medium sized multimedia sessions, we set the average request hold time to be 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.

5.2.1 The Best UF Policy

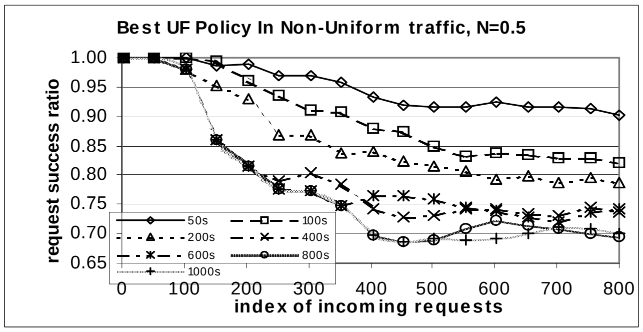
The performances of Best UF policy under different information update frequencies is depicted in Fig 5.a. Given near current system state information, the Best UF policy is a variation of online algorithms that optimize the current assignment (i.e. maximize overall resource utilization) without knowledge of future requests. We examine the performance of Best UF in near current system state information, i.e., with very short update periods. In Fig [5].c we show 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. . This results in a higher utilization of the network and server resources which can be observed from Fig [5].b. 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 resulting in quickly congesting these nodes and links. This causes the network or server to reject the assignment chosen by the Directory Service. Fig[5].b shows that request rejections cause the overall system utilization to fall; the drop in utilization is recorded in the DS during an update; the lightly loaded paths and servers are quickly filled by the DS resulting in more rejections. This is reflected in the rejection pattern graph with update period 1000s, Fig [5].d. In this case, the outdated information causes the directory service to repeat the same assignments, resulting a high rejection ratio from the network and server resources. In our specific experiment settings, server rejection appears dominating the overall performance, because the servers resources are fewer than network resources and are quicker to be saturated.

5.2.2 The Shortest Hop Policy

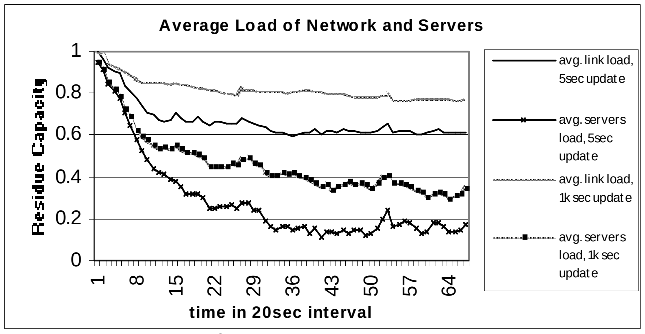
The Shortest Hop Policy is interesting because it provides a shortest widest solution. From the feasible assignment set *X* calculated by base CPSS algorithm, the policy chooses the nearest server from the client in terms of number of hops. If multiple such servers exist, the policy chooses the least loaded one, i.e. the one with least UF value. In general, we can see that the performance of the Shortest Hop Policy is worse than Best UF under the conditions of the experiment (large update period) in Figure [9], which illustrates the comparative performance of the CPSS policies. 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)*= $\sum UF(l), l \in p$. So this makes the shortest widest path a strong candidate to be selected in Best UF policy. The shortest hop policy only considers the length of the path and thus tries to optimize the usage of network resource by using the least number of network links without

[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.

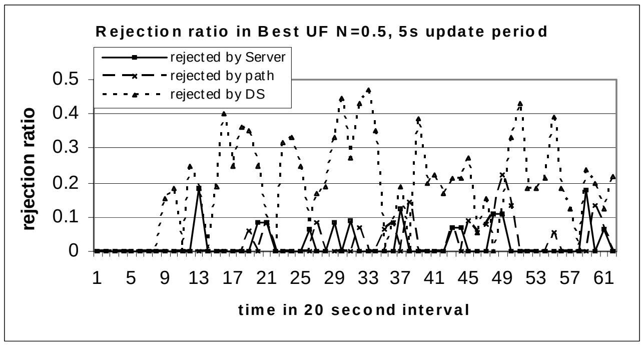
considering current load situations on the links or servers. When state information is not current, as is the case with a large timer, the shortest hop policy tends to initiate congestion earlier because it potentially overloads some paths and servers that are already very congested. Hence, a large number of request rejections in the Shortest Hop policy result from path rejections in the network Fig[6].a With a large update timer, the best UF policy will always route the request along less loaded paths to less loaded servers, and hence the onset of congestion is expected to be slower. This behavior is confirmed by the reject pattern in the 20 second detail graph of 1000s update period. Initial path rejections are caused in the Shortest Hop policy while the Best UF policy initiates the network path rejections later in time. After the update at 1000s, the Shortest Hop policy again exhibits a large number of path rejects; in addition, there are server rejects due to server saturation.



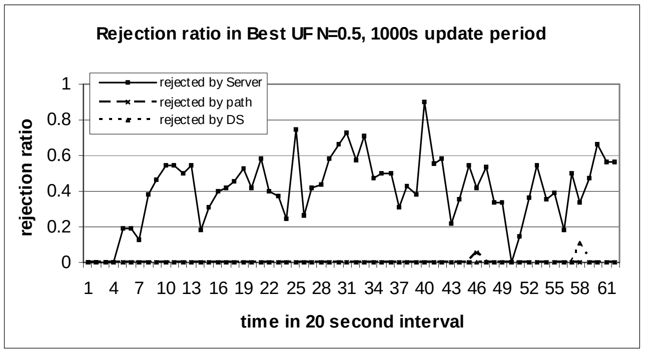
A. Best UF policy with different update periods



B. average load of network and servers



C: rejection pattern when update period equals 5 sec.

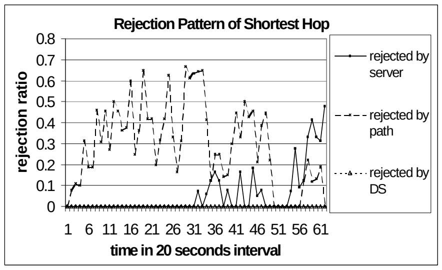


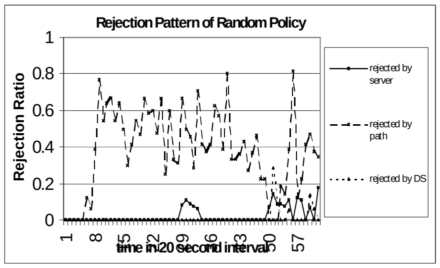
D: rejection pattern when update period is 1000sec.

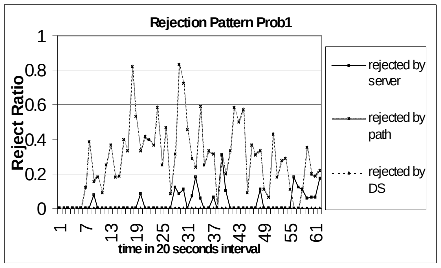
Fig 5: ***Best UF Policy in Non-Uniform Traffic Environments***

5.2.3 The Random Policy

The random policy tries to avoid the oscillation of static polices and balance the load of the system by randomly picking one choice from the feasible assignment set X calculated by CPSS. The random policy does balance the



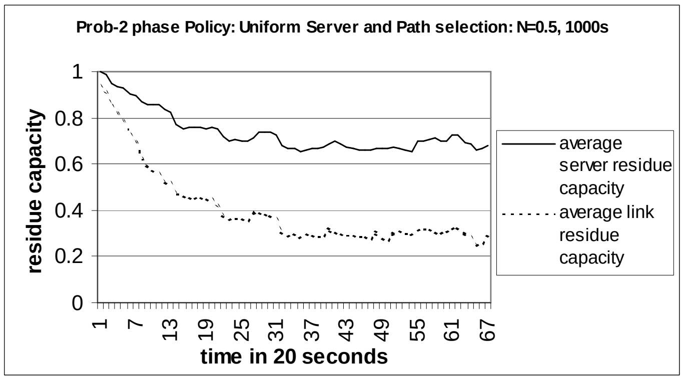


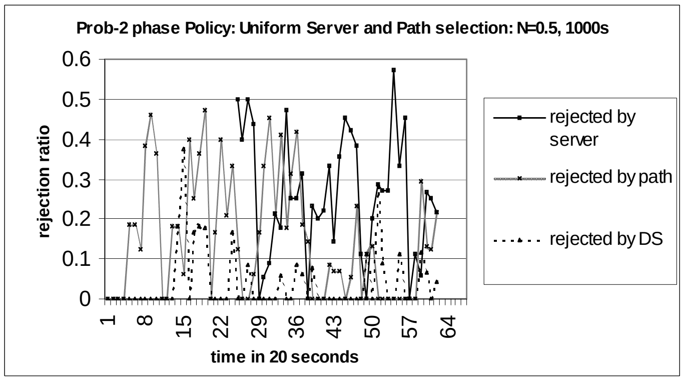


A: Shortest Hop Policy B: Random Policy C: Prob-1ϕ Policy

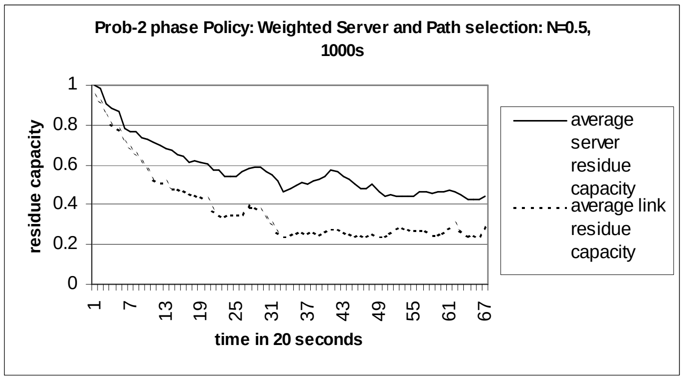
Fig 6: ***Rejection Pattern for CPSS policies with very infrequent sampling. (once per 1000 sec.)***

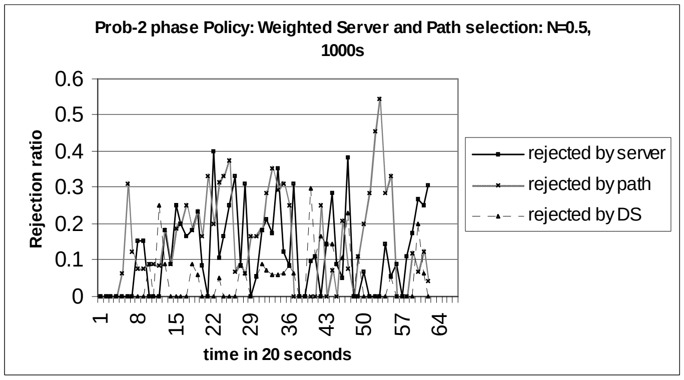
load between the servers, but because it doesn't use the load information of the assignments, it often results in longer network paths and hence the request quickly gets rejected by the network nodes. Fig [6].b shows that the path rejection dominates the overall request reject ratio. This is because the random policy does not differentiate between all the feasible assignments in the set X, repeatedly picking some feasible but marginal assignment, leading to congestion. In summary, the random policy performs consistently worse than the Best UF and Shortest Hop policies.





A and B: Servers weighted equally, update period is 1000s. A: (left), average system load, B: (right), request rejection pattern





C and D: Servers are weighted according to their UF value, update period is 1000s. C: (left) average system load, D: (right) request rejection pattern

Fig 7: ***The Prob-2 ϕ Policy In Non-uniform Traffic Environments***

5.2.4 The Weighted Differential Probabilistic Policy(Prob-1 ϕ):

With the Prob-1 ϕ policy, 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 (see Fig [6].c for its rejection pattern, and Fig [9] for its overall performance). This side-effect is eliminated in the second phase of the Prob-2 ϕ policy where all paths leading to that server are weighted according to their residue capacity.

5.2.5 *The Load Sensitive Differential Probabilistic Policy (Prob-2ϕ):*

Prob-2ϕ policy further improves on the performance of Prob-1ϕ. 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 [Figure 7]. The Prob-2ϕ policy uses either a weighted or uniform (i.e. weight all servers 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; however, they can cause further congestion over a period of time. The uniform policy will not alter the existing load conditions dramatically resulting in more rejects in the short term. The first set of graphs (Figure 7 A and B) corresponds to the uniform policy and the second set (Figure 7 C and D) corresponds to the weighted policy. In the first graph, the relative loads of the server and the network do not change much over the entire duration, with the servers being consistently more loaded than the network. The instantaneous 20 second rejection pattern indicates that the server and network rejects alternate and are comparable in number. In the second graph, the weighted policy tends to pull the two resource utilization levels closer resulting in better balance.

5.3 Information Update

To further evaluate the performance and efficiency of our previous CPSS policies, we compare them with a static "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[9] shows that in non-uniform 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 as seen in Fig [3]. The update period dominates the information accuracy in the directory and thus influences the performance of the CPSS algorithm. In the following we focus on the influence of the information update overhead on CPSS policies

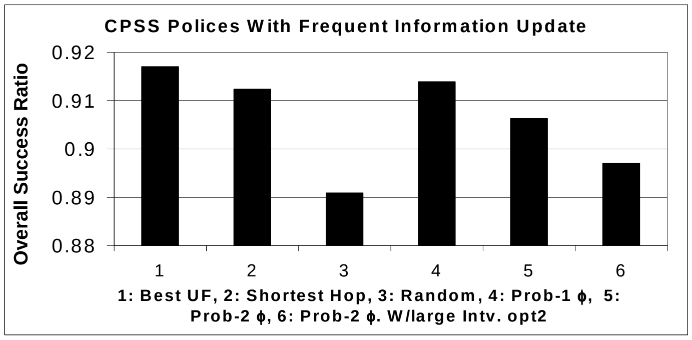


Fig 8: ***CPSS Policies with Frequent Information Updates (period= 5 sec)***

Snapshot Based Information Update: The snapshot based information update can be regarded as a special case of interval based update by setting the range to 1. Our simulation shows that the interval-based policies do not have a significant influence when used together with deterministic CPSS policies. Thus, we focus our discussion of snapshot based information update with deterministic CPSS policies. 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 (600s). Update period values larger than 600s show negligible effects on the performance of CPSS policies.

If the system state information is updated very frequently, our results show that Best UF policy is superior Fig[8]. This is because the Best UF policy tries to find an optimal tradeoff between the shortest and widest paths (See 5.2.2). 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 non-deterministic policies, Random, Prob-1ϕ, and Prob-2ϕ, distribute load between multiple feasible servers and network paths, thus prevent the oscillation and improve 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 the worst.

Interval Based Information Update: In order to further save the cost of information update, we introduced interval based information update policies; the reduction in message overhead cost can be seen from Fig[3]. 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 the interval-based update policies when used together with probabilistic policies, i.e. Prob-2ϕ. Figures [9.B,D,F] show the performance of interval based update policies in various traffic environments.

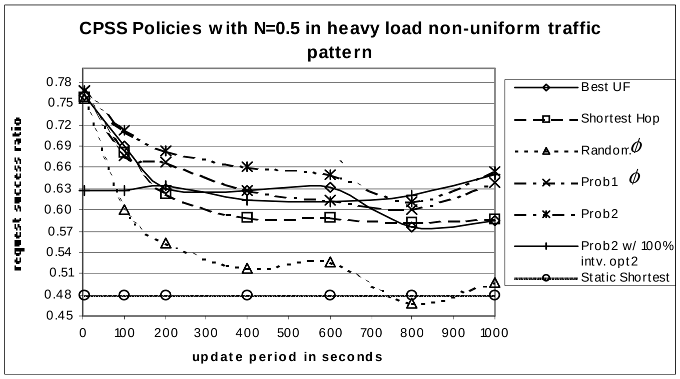
With frequent updates, a smaller range for the interval (50% of the maximum request size) brings better CPSS performance than a bigger interval(100% of the maximum request size). For a large update period, the interval based policies are attractive because we believe it is natural to represent a residual value using a range (instead of an instance value over a long period of time). For a larger update period, a bigger range brings better CPSS performance. This is more obvious in the uniform traffic pattern as shown in Fig [9.B]. The reason is that when the update period is short, representing a residue value using a big range introduces information inaccuracy, a shorter range is better. However, when the update period is very long, a residue value represented using a small range often gets out-dated sooner than a larger one, resulting in a more inaccurate system state information. An analysis of the variations of the interval based update shows the following results. In a lightly loaded non-uniform traffic environment, the OPT2 variation performs better than other variations, i.e. OPT and PESS (Fig [9.c]). In a heavily loaded non-uniform traffic environment, the variations of interval based update do not exhibit an obvious influence on the performance of Prob-2ϕ (Fig[9.e]). However, in general, the performance of Prob-2ϕ is better than other CPSS policies with either interval based or snapshot information update.

5.4 Performance Summary

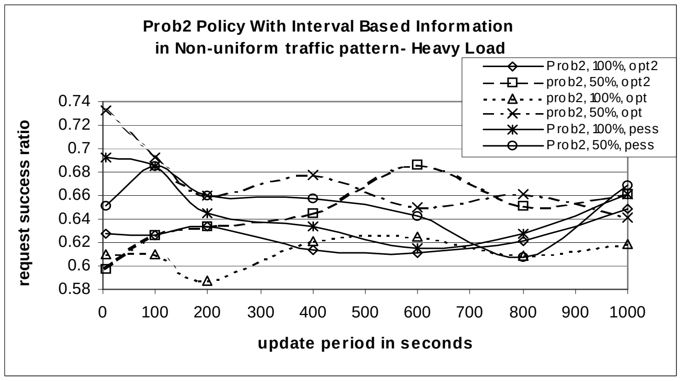
Our evaluation indicates that CPSS based policies perform about 20-30% better on average than non-CPSS policies where server selection is performed using a static, nearest-server policy. The performance of the CPSS policies explored is sensitive to the frequency of update of system state information. With a short update timer, state information is up-to-date, Best UF and Shortest Hop policies result in near-optimal assignments because of CPSS calculation. However, the Best UF Policy performs better than the shortest hop under a large update timer. In general, with a large update period, deterministic policies suffer from oscillation that introduces “hot spots” into the environment causing congestion. The random policy performs a lot worse than other policies consistently. The probabilistic policies (Prob-1ϕ and Prob-2ϕ) perform consistently better with a large update timer. In general, deterministic policies like Best UF normalize server and link loads into one unified utilization measure – thus making it more difficult to handle situations where network and server conditions vary dramatically. Such dramatic variations are better accomodated by probabilistic policies like the Prob-2ϕ (load sensitive differential policies) that treat network and server loads independently in the decision-making process.

Snapshot based update mechanisms introduce larger overhead while no significant performance gains can be observed. Our study indicates that range based update mechanisms are more cost-effective in QoS-sensitive dynamic environments. Furthermore, our experiments reveal that the load-sensitive differential probabilistic policy (Prob-2ϕ) policy performs significantly better under both snapshot and interval based update techniques than other CPSS policies. The variations of the interval based update mechanisms (PESS, OPT and OPT2) exhibit varying performance with OPT2 performing better on average with a large update timer.

A-B: In heavily loaded non-uniform traffic environment

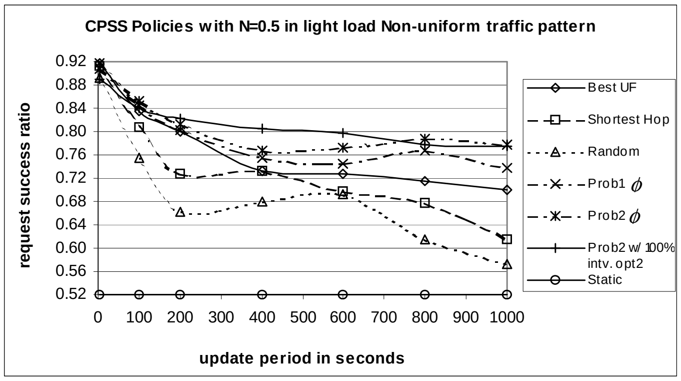


A: Overall performance results.

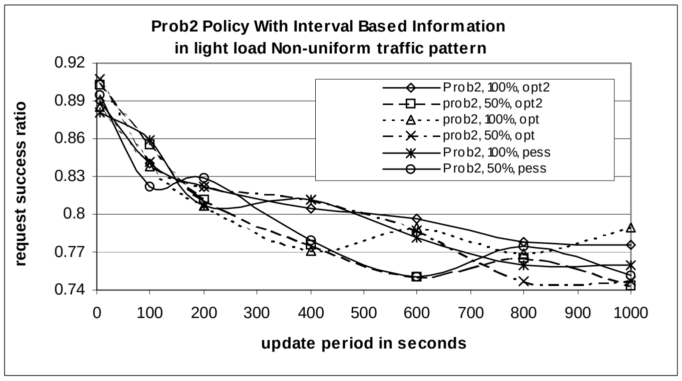


B: interval based update policies with Prob-2 ϕ

C-D: In lightly loaded non-uniform traffic environment

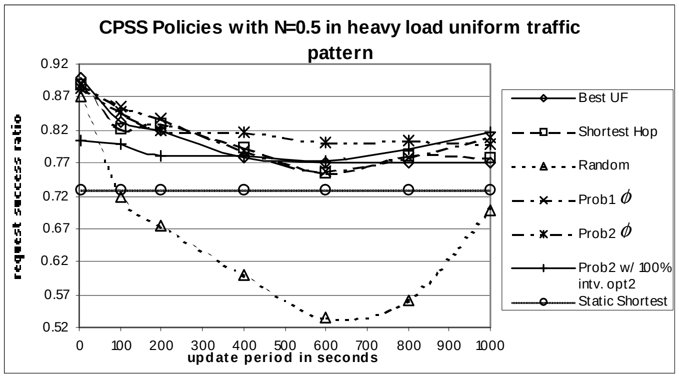


C: Overall performance results.

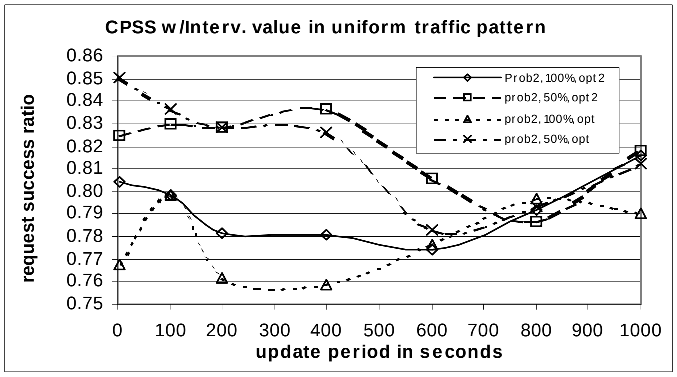


D: interval based update policies with Prob-2 ϕ

E-F: In uniform traffic pattern



E: Overall performance results.



F: interval based update policies with Prob-2 ϕ

Fig 9: *Overall Performance of CPSS with information update*

6 RELATED WORK

QoS-based resource allocation has been addressed independently within the networking, multimedia and distributed computing communities. We specifically focus on two areas – i.e. replicated service selection and QoS routing. Static nearest server algorithms [GS95] 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 resulting in performance degradation in some situations. On the other hand, dynamic server selection policies have been studied in the context of replicated web services [FJPZGJ99,MDZ99,CC97,FBZA97] and of multimedia streaming applications [VR97,HCM98]; both of them did not explicitly address the load factor of the underlying network.

QoS-based routing techniques have been explored in depth [CN99,CRS97,BS98,BEZ93]. QoS-based extensions of the traditional Bellman-Ford/Dijkstra algorithm [BG92] incorporate an available bandwidth parameter and use widest-shortest path [GOW98,CN98] calculations. Experimental efforts studying the combined performance and overhead cost of different information collecting and routing policies [AGKT98] 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 [LO98] 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[H92]. In this paper, we apply these techniques to address a server scheduling problem where the load balancing for both computational and network resources is considered in an unified framework.

7 CONCLUDING REMARKS

In this paper, we have demonstrated the effectiveness of an integrated approach to QoS-based resource provisioning in a distributed environment. Prior work has categorized functionality that must be addressed at different levels in the system. For example, routing issues are dealt with in the network layer and server load balancing issues are handled in a distributed computing layer. Emerging distributed applications require end-to-end service guarantees that must be enforced at multiple levels; a more holistic approach to resource allocation is required. The proposed middleware solution uses a directory service component that maintains reasonably accurate information of the network and server system state using which 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 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 and adaptive dynamic range based techniques [FV01-1]. 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 [FV01-1,FV01-2]. 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[WCDJM98,W95,MV02].

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) [HV02-2,HV01-1]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 in 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

[AGKT98] G.Apostolopoulos, R.Guerin, S.Kamat, S.K.Tripathi, “Quality of Service Routing: A Performance Perspective”. *In Proc. of ACM Sigcom’98*
[BG92] D. Bertsekas and R. Gallager. “Data Networks”, *Prentice Hall, Englewood Cliffs, N.J., 1992*
[BS98] Lee Breslau and Scott Shenker, “Best-Effort versus Reservations: A Simple Comparative Analysis”, *Sigcomm ’98*
[BZBHJ97]R. Braden and L. Zhang and S. Berson and S. Herzof and S. Jamin, “Resource Reservation Setup Protocol (RSVP) -- Version 1 Functional Specifications”, *RFC 2205, Septemeber 1997.*
[CC97] R.L.Carter and M.E.Crovella, "Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks", *in Proceedings of Infocom '97.*
[CN98] S.Chen, K.Nahrstedt, “On Finding Multi-Constrained Paths”, *in Proceedings of IEEE ICC 98, June 1998, Atlanta.*
[CN99] S.Chen and K.Nahrstedt, “Distributed Quality-of-Service Routing in Ad-Hoc Networks”, *IEEE Journal on Special Areas in Communications, Special Issue on Ad-Hoc Networks, 1999.*
[CRS97] I.Cidon, R.Rom, and Y.Shavitt. Multi-path routing combined with resource reservation. Infocom’97
[FBZA97] Zongming Fei, Samrat Bhattacharjee, Ellen W.Zegura, Mostafa H.Ammar, A Novel Server Selection Technique for Improving the Response Time of a Replicated Service, Infocom ‘97
[FJPZGJ99] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, Y. Jin. “An Architecture for a Global Internet Host Distance Estimation Service”. *INFOCOM '99, March 1999.*
[FV99] Z.Fu, N.Venkatasubramania. “Combined Path and Server Selection In Dynamic Multimedia Environments”, *ACM MM ’99*
[FV01-1] – “Directory Based Information Collection for QoS Provisioning in Dynamic Multimedia Environments”, *IEEE International Conference on Distributed Computing Systems (ICDCS 2001).*
[FV01-2] – “An Evaluation of Composite Routing and Scheduling Policies for Dynamic Multimedia Environments”, *IEEE IPDPS 2001.*
[GOW98] Roch Guerin, Ariel Orda, and Douglas Williams, “QoS Routing Mechanisms and OSPF Extensions”,*Globcom’98*
[GTE99] GTE Internet Access Services: Managed Hosting with Traffic Distributor, http://www.bbn.com/services/hosting/traffic
[GS95] J.D.Guyton and M.F.Schwartz, “Locating nearby copies of replicated internet servers,” in ACM SIGCOMM,1995
[H92] R.Hassin. “Approximation schemes for the restricted shortest path problem”. *Math. Of Op. Research, 1992.*
[HCM98] Mor Harchol-Balter, Mark E.Crovella, and Cristina D.Murta, “On Choosing a Task Assignment Policy for a Distributed Server System”, *In Proceedings of Performance Tools ’98, Lecture Notes in Computer Science, Vol. 1468, pp. 231-242, Sep 1998.*
[HV01] Qi Han and Nalini Venkatasubramanian. "*AutoSeC: An integrated Middleware Framework for Dynamic Service Brokering*", *IEEE Distributed Systems Online 2001, Vol. 2, No. 7, 2001.*
[HV02] Qi Han and Nalini Venkatasubramanian. “*A Cost Driven Approach to Information Collection for Mobile Multimedia Environments*.” *IEEE Intl Conf on Mobile and Wireless Communications Networks, MWCN 2002.*
[LO98] D.H.Lorenz and A.Orda. QoS “Routing in Networks with Uncertain Parameters”. *INFOCOM ’98*
[MDZ99] Andy Myers, Peter Dinda, Hui Zhang, “Performance Characteristics of Mirror Servers on the Internet”, *Globecom ’99*
[MV02] Shivajit Mohapatra and Nalini Venkatasubramanian. “*PARM: Power-Aware Reconfigurable Middleware*”, Submitted for publication.
[VDMW01] Nalini Venkatasubramanian, Mayur Deshpande, Shivjit Mohapatra, Sebastian Gutierrez-Nolasco and Jehan Wickramasuriya, “*Design & Implementation of a Composable Reflective Middleware Framework*”, *IEEE ICDCS 2001.*
[VR97] N.Venkatasubramanian and S.Ramanathan, " Load Management for Distributed Video Servers", *Proceedings of IEEE ICDCS '97, May 1997.*
[WCDJM98] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, G. Mendez, "Cost and Imprecision in Modeling the Position of Moving Objects", *Intl. Conference on Data Engineering, 1998.*
[W95] O. Wolfson, P. Sistla, S. Dao, Kailash Narayanan and Ramya Raj. “ View Maintenance in Mobile Computing”. *SIGMOD RECORD, Dec 1995, Invited Article.*

APPENDIX I– FEASIBILITY PROOF OF CPSS

Lemma 1: If path P_n, P_n =*{(O,*$u_{n,1}$*),(*$u_{n,1}, u_{n,2}$*),…,(*$u_{n,k-1}, u_{n,k}$*),(*$u_{n,k}$ *,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 $\forall(u,CD)\in E'$, u is a server node. This shows that there is at least one server node on P_n, and on the path *{(O,*$u_{n,1}$*),(*$u_{n,1}, u_{n,2}$*),…,(*$u_{n,k-1}, u_{n,k}$*),(*$u_{n,k}$ *,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 don't have outgoing edges when constructing the graph G' and this is impossible. This proves the Lemma. *QED*

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) ,…,($u_{k-1,} s_i$) }, satisfies the feasibility condition if $DIST_{CD}[d] < \infty$, $d \le DL_r$, and $P_d = p_i \cup (s_i, CD)$, where P_d is the corresponding feasible path with delay d.

Proof: If $DIST_{CD}[d] < \infty$, 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 condition (1),(2) is 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 condition. 2) Feasibility condition 3 is satisfied because $d < DL_r$, $DIST_{CD}[d] < \infty$ and $Delay(P_d) = EED^{X_i^d} = DL^{p_i} + RSP^{s_i} \le DL_R$. *QED*

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 edges of a vertex for each delay constraint value. So the total time complex is *O(*$|DL_r|$*E').*