QoS-based Resource Discovery in Intermittently Available Environments

Yun Huang and Nalini Venkatasubramanian
Dept. of Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
{yunh, nalini}@ ics.uci.edu

Abstract

In this paper, we address the problem of resource discovery in a grid based multimedia environment, where the resources providers, i.e. servers, are intermittently available. Given a graph theoretic approach, we define and formulate various policies for QoS-based resource discovery with intermittently available servers that can meet a variety of user needs. We evaluate the performance of these policies under various time-map scenarios and placement strategies. Our performance results illustrate the added benefits obtained by adding flexibility to the scheduling process.

1. Introduction

The evolution of the Internet and differentiated services has enhanced the scope of applications that can execute in global information infrastructures. We are observing a dramatic increase in the amount of multimedia content being delivered over the Internet today. This is fueled by multimedia-enhanced applications and websites e.g. distance learning, video-conferencing, news-on–demand etc. Such applications are resource intensive, and consume significant network, storage and computational resources. Furthermore, multimedia applications also have Quality-of-Service (QoS) requirements that specify the extent to which properties such as timeliness can be violated. Efficient and adaptive resource management mechanisms are required to ensure effective utilization of resources and support an increasing number of requests.

Global grid infrastructures [14, 2] allow the use of idle computing and communication resources distributed in a wide-area environment; a properly managed grid system can form the infrastructure upon which multimedia applications can be executed. In fact, applications that provide delivery of multimedia information are especially well suited for grid based environments; often, the content delivered is read-only, and coherence of multiple replicas is not an issue. However, systems on a computational grid are available only intermittently.

We define "intermittently available" systems as those in which servers and service providers may not be available all the time, e.g. grid-based systems might provide resources/services for only a few hours a day. Effective load management in an intermittently available multimedia environment requires:

(a) resource discovery and scheduling mechanisms that will ensure the continuity of data to the user while servers are intermittently available.
(b) data placement mechanisms to ensure that popularly requested information is replicated such that the data is always available.

In this paper, we focus on the first of these two problems. Using a graph-theoretic approach, we propose a generalized resource discovery algorithm that can determine suitable scheduling plans for incoming multimedia requests. User requests vary widely in the immediacy of the response (startup latency), continuity needs of the applications and resource characteristics of the client end systems. We propose and evaluate a family of policies that can cater to the various user requirements discussed above.

The rest of this paper is organized as follows. We illustrate the system architecture in Section 2. Section 3 introduces a generalized version of the Discovering Intermittently Available Resources (DIAR) algorithm in such an intermittently available environment. Section 4 details DIAR policies for satisfying various QoS requirements. We evaluate the performance of the proposed techniques in Section 5 and conclude in Section 6 with future research directions.

2. System Architecture

The system architecture is depicted in Fig 1. The system consists of clients and multimedia servers distributed across a wide area network. The distributed servers provide resources to store the multimedia data that can be streamed or downloaded to clients at suitable QoS levels. The resources provided include high capacity storage devices (e.g. hard-disks), processor, buffer memory, and high-

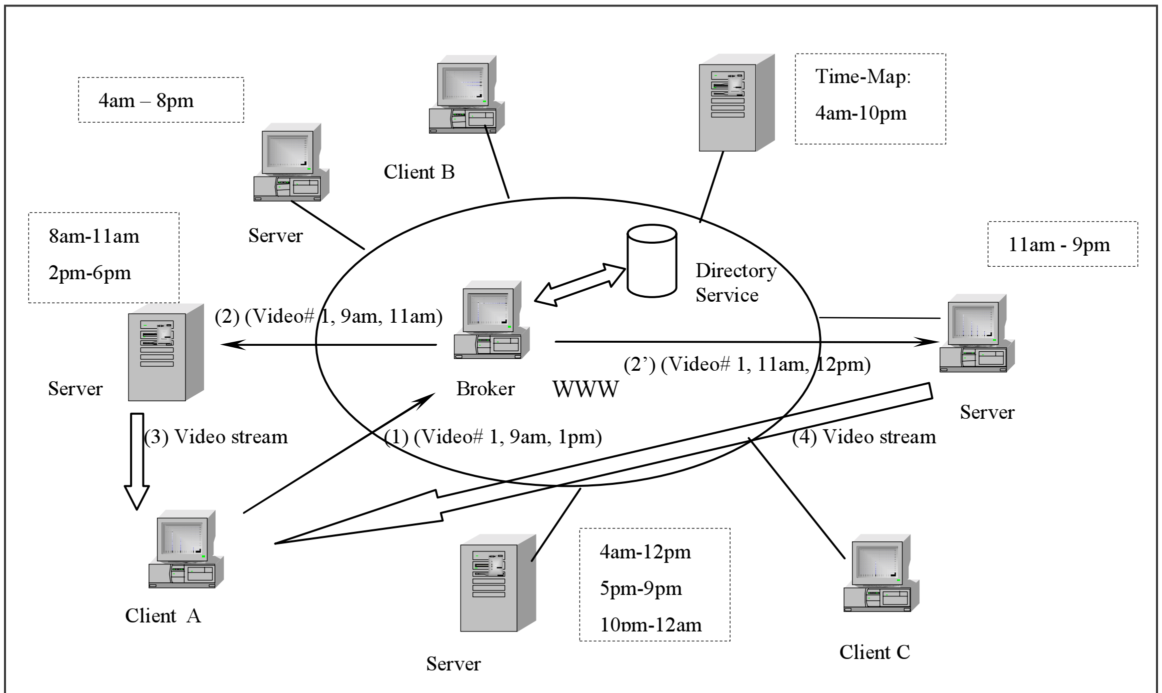

Fig 1. System Architecture: the arrows indicate the processing of a request from client A.

speed network interfaces for real-time multimedia retrieval and transmission. Servers may be intermittently available and the time map of servers containing available server times is predetermined. However, the availability of resources on servers can vary dynamically. To accommodate a large number of video objects, the environment includes tertiary storage[1].

Clients issue requests for multimedia information that may be replicated across servers. Client requests can vary significantly; hence requests are parameterized to specify the data resource requirements and service characteristics – i.e. earliest start time, latest finish time, service duration, and whether the service required is continuous or discontinuous etc. The key component of the infrastructure is a ***brokerage service***. All incoming requests from clients are routed to the broker that determines whether or not to accept the request based on current system conditions and request characteristics. Specifically, the broker:

(a) discovers the appropriate set of resources to handle an incoming request;
(b) coordinates resource reservation and schedules these requests on the selected resources.

Several kinds of information are required to ensure effective resource provisioning – this includes server resource availabilities, server time maps, replica maps, network conditions etc. This information is held in a directory service (DS) that is accessed and updated suitably by the broker. Using state information in the DS, the broker determines a candidate sever or a set of servers that can satisfy the request. Once a solution for the incoming request (i.e. scheduled servers and times) has been determined, the broker will update the directory service to reflect the allocated schedule. The goal is to improve the overall system performance, in order to facilitate more requests. If the broker determines that more than one server must provide the desired service, it transforms the single user request into multiple requests that will execute on the different servers. The service provided may be continuous or discontinuous; and can be finished without or with some delay. In the typical mode of operation, the servers guarantee the provisioning of appropriate QoS to each request, once resources have been reserved.

3. Discovering Intermittently Available Resources – The DIAR Algorithm

In this section, we present a generalized algorithm for discovering intermittently available resources (DIAR) by modeling the DIAR problem using a graph-theoretic approach. We develop a generalized solution based on a network flow analogy; further refinements will be discussed in the next section.

[1] For the purposes of this study, however, we do not model dynamic transfer from tertiary storage.

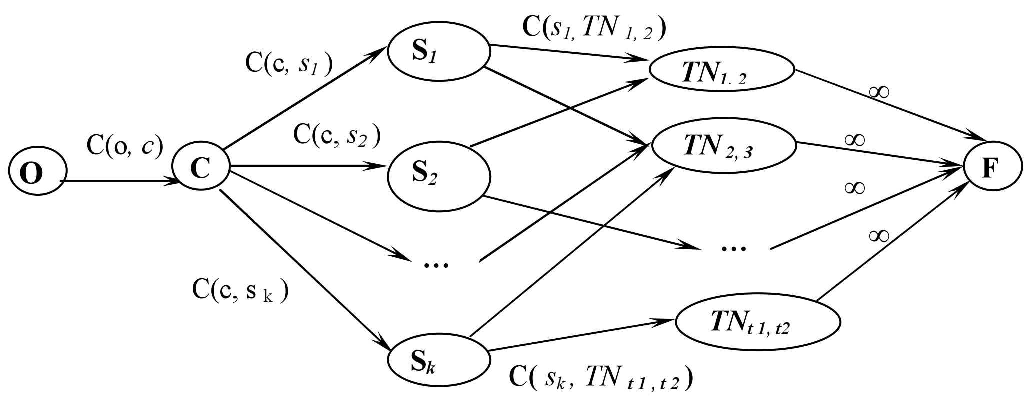

Fig 2. A Graph-based representation of the DIAR algorithm

3.1. Modeling the DIAR problem as a Network Flow Diagram

Without loss of generality, we assume that the incoming request is for a video object. We model the incoming request R from the client as:

R: < VID_R, ST_R, ET_R, $Type_R$, QoS_R >

Where VID_R corresponds to the requested video ID; ST_R is the request start time; ET_R is the end time by which the request should be finished; $Type_R$ represents the type of the request including (a) immediacy of the response (b) whether the request must be continuous or can be discontinuous and (c) whether the request must be executed on a simple server or may be executed on multiple servers. QoS_R represents the QoS parameters of the request, i.e. the resources required by the request and the duration for which these resources are needed. We model the resources needed by a request using 4 parameters: the required disk bandwidth (R_{DBW}), required memory resource (R_{MEM}), required CPU (R_{CPU}) , and the required network transfer bandwidth (R_{NBW}); D_v represents the duration for which these resources are required. The above QoS parameters may be assumed to be defined for each video and may therefore be known by the broker ahead of time. Note that since many requests can obtain service from the same copy of the video object, the actual storage (i.e. disk space) occupied by an object is not relevant to the scheduling process; the storage bandwidth is consumed on a per request basis and is the relevant factor that must be considered.

We describe each server *s* with the following configuration: server time map (STM_s) indicating the times at which a server is available and a server resource map (SRM_s) indicating server resources available at the different times. For simplification, we divide the time of a day into fixed size units, (for example, 24 units to present 24 hours of the day). We represent the available resources of a server *s* at a particular time *t* using four parameters [20]: CPU cycles ($S_{AvailCPU}(t)$), memory buffers ($S_{AvailMEM}(t)$), network transfer bandwidth ($S_{AvailNBW}(t)$), and disk bandwidth ($S_{AvailDBW}(t)$). Hence $SRM_s(t)$ = ($S_{AvailCPU}(t)$, $S_{AvailMEM}(t)$, $S_{AvailNBW}(t)$, $S_{AvailDBW}(t)$).

In order to deal with the capacity of each server over time in a unified way, we define a Load Factor (*R, S, t*) for a request r on server s at time *t*, as

Load Factor (*R, S, t*)
$= \text{Max}\,[\,CPU_a, MEM_a, NBW_a, DBW_a\,]$
$CPU_a = R_{CPU} / S_{AvailCPU}(t)$
$MEM_a = R_{MEM} / S_{AvailMEM}(t)$
$NBW_a = R_{NBW} / S_{AvailNBW}(t)$
$DBW_a = R_{DBW} / S_{AvailDBW}(t)$

Thus, the Load Factor [16] of a server *s* at time *t*, LF(*R, S, t*), is determined by the bottleneck resource at time *t*. Furthermore, the order in which we explore potential servers uses a worst-case assumption on the load-factor over all time units (between ST_R and ET_R) during which the server is available. For example, if the granularity of the time units in a day is 24 (24 hours/day), and ST_R is 5am and ET_R is 10am, then we consider

$LF(s) = \text{Max}(LF_5, LF_6, LF_7, LF_8, LF_9, LF_{10})$.

A more optimistic approach would be to use the average load-factor over the time period. Since we have information about future reserved resources on a server, it is possible to estimate the load-factor based on existing system conditions and future scheduled requests.

We model the DIAR algorithm as a directed graph G<V, E> (see Fig 2). Nodes O and F are artificial nodes and represent the source vertex and sink node respectively [11]. C represents the client, while S_j represents server *j*, ($1 \le j \le N$, if there are N servers). We introduce a set of time nodes, TN that represent the entire time period. Each time node $TN_{t1,t2}$ represents the period of time from *t1* to *t2*, [*t1, t2*).

```
DIAR (G<V, E>, R: < VID_R , QoS_R , ST_R , ET_R , Type_R> {
    /* Initialization:*/
    (1)  for each edge $(u , v) \in E$
    (2)  $f(u , v) \leftarrow 0$  /* $f(u , v)$ is the flow of the edge (u, v) */
    (3)  $f(v , u) \leftarrow 0$
    (4)  /*Reduce the graph:*/
         Eliminate infeasible servers that can not provide the service for VID_R ;
         Eliminate irrelevant time nodes that can not provide service during the request period.
         (i.e. $ET_R \leq t1$ or $t2 \leq ST_R$)
    (5)  /*ordering the server and time nodes: */
         Reorder the left server nodes of $S_i$ and $S_j$, so that e.g. $i \leq j,\ LF(i) \leq LF(j)$ .
    (6)  define the weight of each edge $(u , v) \in E$, such as:
           • C(o, c) = $D_v$, (duration of the requested video);
           • C(c, $s_j$) = the total amount of service time units that server j can provide;
           • C($s_j$, $TN_{t1,t2}$) = t2-t1, if server j can provide service in the time period [t1, t2);
           • C($TN_{t1,t2}$ , F) = $\infty$.

    /* Main Loop */
    (7)  while there exists an Augmenting Path $p$ [11] from $O$ to $F$ in $G$:
    (8)        C(p) $\leftarrow$ min (C(e) | $e \in p$ )
                        /* e is the edge of p, C(e) is the weight of edge e.*/
    (9)        for each edge $(u , v) \in p$
    (10)             $f(u , v) \leftarrow f(u , v)$ + C(p)
    (11)             $f(v , u) \leftarrow -f(u , v)$
}
```

Fig 3. Discovering Intermittently Available Resources Algorithm

We assign directed edges connecting these nodes with weights as follows:

- C(o, *c*) = D_v, (duration of the requested video).
- C(*c*, s_j) = the total amount of service time units that server *j* can provide within the request time period.
- C(s_j, $TN_{t1,t2}$) = *t2-t1,* if server *j* can provide service in the time period [*t1, t2*). For simplicity, we also assume that all time nodes represent equal durations (one hour in our simulation).
- C($TN_{t1,t2}$, *F*) = ∞.

Thus, each edge has a nonnegative weight. If we treat this graph as a Network Flow Graph, the DIAR problem can be cast as a maximum flow problem, i.e. find the maximum network flow $|f| = C(o,c)$. The flow |*f*| is a measure of the maximum weight along paths in a weighted, directed graph [1].

The feasibility condition: An *Augmenting Path* [11] $p(S_j, TN_{t1,t2})$ from ***o*** to ***F*** is a path with nonnegative weight on the edges along the path. An augmenting path is feasible if and only if it satisfies all the following conditions:

- $VID_R \in$ the set of videos for which the server *j* can provide service.
- $S_{Avail\ NBW} \geq R_{NBW}$, $S_{Avail\ CPU} \geq R_{CPU}$, $S_{Avail\ MEM} \geq R_{MEM}$, $S_{Avail\ DBW} \geq R_{DBW}$, (The available resources can meet the resource requirements of this video).
- $ST_R \leq t1$, $t2 \leq ET_R$.

We define a feasible set X_p as a set of all the augmenting paths that meet the feasibility conditions:

$$\sum C(S_j, TN_{t1,t2}) \geq C(o,c)\ ;\ \text{for } \forall (S_j, TN_{t1,t2}) \in X_p .$$

This corresponds to a feasible scheduling solution of this video request.

Optimality of DIAR: Given a client request R: <VID_R , QoS_R , ST_R , ET_R , $Type_R$>, a solution X_p is optimal if and only if it satisfies the feasibility condition and a policy dependent optimality criterion. Examples of optimality criteria include minimum number of servers, least-load factor, earliest-starting-time, earliest-finishing time, most continuous service, etc. The policies defined later in the paper use two of the above criteria that we believe are highly relevant for the proposed environment, i.e. least Load Factor and earliest-starting time.

3.2. The DIAR algorithm.

The DIAR algorithm uses well known network flow techniques (e.g. Ford – Fulkerson Algorithm [11]) to determine the Maximum flow $|f|$ (see Fig 3).

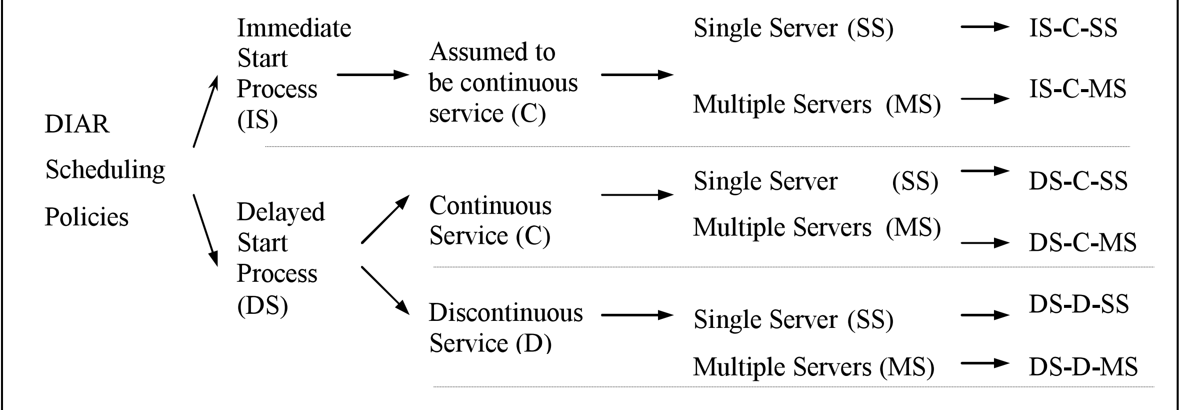

Fig 4 DIAR Scheduling Policies

The methods used for searching the augmenting path decide the characteristics of the final result. If the request must be executed by a single server, the search will start from the server nodes to determine the first server that can provide service for the entire time duration, which ensures the least Load Factor criterion. On the other hand, if the request can be finished by multiple servers, we will search the augmenting path by the time nodes; this will implement an earliest-starting time policy.

If the maximum flow $|f|$ of the graph equals $C(o,c)$, the flows we get are correspondent with a schedule solution we can use to accept this request. In the following section, we describe several policies to determine a suitable augmenting path according to request characteristics. If N is the number of server nodes in the order of increasing Load Factor, and the T is number of time nodes, the complexity of DIAR will be *O(N*T)*.

4. A Family of DIAR policies

In this section, we describe the different DIAR policies to satisfy various QoS requirements of multimedia requests. We classify the policies based on three criteria:

(a) whether the request must be immediately started with zero startup latency (IS) or if the request startup can be delayed (DS);

(b) whether the request must be continuously (C) processed until completion or if the request can be discontinuously (D) executed;

(c) whether the service is provided by one single server (SS) , or if the service can be provided by multiple servers (MS).

The classification of DIAR policies is illustrated in Fig 4. Six policies are proposed: IS-C-SS, IC-C-MS, DS-C-SS, DS-C-MS, DS-D-SS, and DS-D-MS[2]. We will use the above nomenclature in the remainder of this paper to refer to the six policies studied. In the implementation of the above policies, the technique used to search for augmenting paths may be server-driven (pick a server first and then determine suitable time nodes) or time-driven (pick a time-node first and then select a feasible server). In general, the single-server policies will be server-driven and multiple server policies will be time-driven.

Immediate Start Continuous Single Server (IS-C-SS): IS-C-SS is the most restrictive of the policies studied and is used to provide continuous service without delay by only one single server. This implies that a video request can be accepted only if it can execute continuously to completion without any delay. Since this policy has such strict requirements, we anticipate that it will exhibit the worst performance in terms of request acceptance. To implement this DIAR policy, we calculate augmenting paths of potential servers in the order of increasing load factor (LF) and stop when we find the first server that can meet the request constraints. If no feasible server is found, the request is rejected.

Immediate Start Continuous Multiple Servers (IS-C-MS): IS-C-MS is similar to IS-C-SS; the only difference is that we can use multiple servers to provide immediate continuous service. We expect that more requests will be accepted with this policy in comparison to IS-C-SS since there is a wider choice of possible servers. In order to realize this policy, we search for augmenting paths in the DIAR network flow graph for each time node and pick the least loaded server within each time period. The request will be rejected only if there is no feasible server for any time node within the duration of service.

Delayed Start Continuous Single Server (DS-C-SS): The DS-C-SS policy permits a startup delay as long as

[2] We ignore the cases where immediate start requests are discontinuous since many applications that have zero startup latency requirements also require continuous service.

service is completed before the stipulated completion time (ET_R); The search for augmenting paths is server-driven, i.e. we follow the chain of servers beginning with the least loaded server and stop when we find the first server in the sequence that can provide continuous service obeying the completion time constraint. Furthermore, we select the earliest feasible starting time that is available on the server. The request is rejected if there are no feasible servers.

Delayed Start Continuous Multiple Servers (DS-C-MS): DS-C-MS relaxes the single server constraint of the DS-C-CS policy. It permits multiple servers to provide the required service. The search for augmenting paths is time-driven; the least loaded server within each time period is selected. As before, service is initiated at the earliest-feasible time at which a server can provide the requested service. The request is rejected if no server combination exists to provide continuous service.

Delayed Start Discontinuous Single Server (DS-D-SS): In DS-D-SS, the request may execute discontinuously but on a single server. To realize this policy, the search for augmenting paths is server-driven, only if a server can provide the whole period of service time for the request, request will be accepted regardless of whether the service will be continuous or discontinuous. If there is no feasible server, the request will be rejected.

Delayed Start Discontinuous Multiple Servers (DS-D-MS): DS-D-MS can provide delayed service discontinuously by multiple servers. Intuitively, this policy should exhibit the minimum number of rejections as compared to the other policies studied. In the DIAR algorithm, we begin searching for an augmenting path from the time nodes until we find enough time nodes to service the requested video, otherwise, the request will be rejected.

5. Performance Evaluation of the Resources Discovery Policies

In this section, we evaluate the performance of different resource discovery policies under various server resource constraints and time configurations.

5.1. System Model

The basic video server configuration used in this simulation includes 20 data servers and 100 video objects; each server has a storage of 100 GB and network transfer bandwidth of 100Mbps. These parameters will be appropriately altered for simulation studies. For simplicity, the CPU and memory resources of the data servers are assumed not to be bottlenecks. In the following simulations, we also assume that the duration of each video is 3 hours; each video replica requires 2 GB of disk storage; and a network transmission bandwidth of 2 Mbps.

5.2. Time Map and Placement Strategies

The service time map for each server keeps the information of when the server will be available during a day. We study three approaches to model the time map of each server:

- **T1:** *Uniform availability* – All the servers are available (or unavailable) for an equal amount of time; furthermore, the servers are divided into groups, such that within each group, the time distribution covers the entire 24-hour day, as shown in Fig 5. In our simulations, we use the uniform availability with duration = 6 hours as the first time map strategy, T1. We also executed the entire set of experiments with a finer granularity of the time map, with available duration = 3 hours, and the results obtained were similar to the coarse granularity case where duration = 6 hours.
- **T2:** *Random availability* – We randomly choose the number of hours during the day when a server is available. The time at which a server is available is also randomly chosen on a per-hour basis; hence the available times may be continuous, discontinuous or partially continuous.
- **T3:** *Total availability* – All the servers are available all the time.

When using T1 or T2, clustering the servers into

Hour	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Server1	▨	▨	▨	▨	▨	▨																		
Server2							▨	▨	▨	▨	▨	▨												
Server3													▨	▨	▨	▨	▨	▨						
Server4																			▨	▨	▨	▨	▨	▨
…																								

Fig 5. A sample uniform availability Time Map - the above figure illustrates the time map where the size of each server group is equal to 4; and the servers are continuously available for a duration of 6 hours.

"server groups" is an important concept for ensuring that at least one replica of video objects is always available. This implies that there are no "blank" spots in service availability. Note that the placement strategies will take this issue into account.

A data placement model specifies the number of replicas for each video object, and the servers on which the replicas are placed. Placement policies may be statically determined during system initialization or dynamically altered at run-time. Dynamic strategies, e.g. [12], are inherently more adaptive; however in an intermittently available environment such strategies can be quite complex. For instance, requests can reserve server resources ahead of the execution time. This enforces more constraints on the dereplication process since replicas not immediately required (but required in the future) cannot be dereplicated unless there is a guarantee that the replica will be reinstated when required. Optimal placement strategies are beyond the scope of this paper and will be addressed elsewhere; we focus on static placement strategies in our simulations.

We propose deterministic and non-deterministic placement policies to determine which replica is created on which server. We select three widely different placement strategies to evaluate our scheduling mechanisms.

- **P1:** *Group-based equal placement* – Ignore the popularity information of the video objects. Cluster the servers into groups so that the total available service time of each group covers the entire day. Each video object is associated with exactly one group, so that every server in the group has a replica of this video object.
- **P2:** *Popularity-enhanced deterministic placement* – We classify the video objects into two groups, i.e. very-popular and less-popular. A replica of very-popular video objects is placed on every server, assuming resource availability. Less-popular video objects are evenly placed in the remaining storage space. The even placement policy attempts to create an equal number of replicas for less-popular video objects.
- **P3:** *Popularity-based random placement* – Choose the number of replicas for a video object based on its popularity, and randomly distributed these replicas among the feasible servers, that have available disk storage.

In the remainder of this paper, we will use T1, T2 and T3 to identify the three time map strategies; P1, P2 and P3 to identify the three placement strategies.

5.3. Request Model

Since the focus application of this paper is the delivery of multimedia services, we choose an appropriate request model to characterize incoming requests to the broker, i.e., Zipfian law [12], with the request arrivals per day for each video V$_i$ being given by:

$$P_{r.}(V_{iI}\ is requrested) = \frac{K_M}{i}, \text{ where } K_M = \left(\sum_{i=1}^{M}\frac{1}{i}\right)^{-1}.$$

We compute the probability of request arrival in hour j to be: $p_j = c/(j^{1-\phi})$ for $1 \le j \le 24$, where Φ is the degree of skew and is assumed to be 0.8, and $c = 1/(\sum(1/j^{1-\phi})),\ 1 \le j \le 24$. From the request arrivals per day for each video and the probability distribution of requests for each hour, the number of requests that arrive in each hour for each video V$_j$ is computed. The validity of this model is confirmed in the studies of Chervenak [6] and Dan, Sitaram and Shahabuddin [13], whose analyses examined statistics in magazines for video rentals and reports from video store owners. Both of them concluded that the popularity distribution of video titles could be fitted into a zipfian distribution.

5.4. Performance Evaluation

We evaluate system performance with the time map and placement strategies explained above and use the number of rejections (i.e. success of the admission control process) as the main metric of evaluation. We compare how the different DIAR policies perform under varying system configurations (such as server storage) and varying number of data servers. Finally, we determine the tradeoff between the service quality and the system performance.

Request admission performance

We compare the performance of the above six scheduling policies under different placement and time map strategies. Fig 6 represents the number of rejections over the time for the six policies. The title of each graph indicates the placement and the time map strategy used in this simulation environment. The results produced with the random time-map and random placement strategies are averaged over several executions.

Intuitively, when servers are always available, the multiple server cases should always have a better acceptance rate. This is because, in general, the single server case is a constrained version of the multiple server case and therefore has fewer options for resource selection. However, we notice in our experiments that another factor comes into play: i.e. the amount of available resources; and this introduces variation in the outcome. Consider the case of the P2-T3 configuration; the algorithm causes resources in multiple servers to be reserved in the future, hence larger number of resources become unavailable to new incoming immediate-start requests in the next round. This is also true sometimes in P1-T3 and P3-T3, i.e. DS-D-MS exhibits higher rejection rates than DS-D-SS.

In the case of intermittently available servers, the single server case becomes highly restricted since finding a single continuously available server that meets the timing requirements is difficult. The multiple server policy offers much more flexibility, since request execution times can span multiple servers. However, this can cause the non-popular requests to be distributed across servers that contain popular objects and cause early saturation of servers with popular objects, resulting in the rejection of many popular requests. As can be observed, the multiple server policies perform better within the time map T1 and T2. As expected, the most restrictive policy, i.e. IS-C-SS exhibits the worst performance and the most flexible policy DS-D-MS has the smallest number of rejections.

Impact of the service time map

In this section, we analyze the impact of different time maps. In the T3 strategy, servers are available all the time, hence for a given placement strategy, we notice that T3 has the fewest number of rejected requests. This serves as a comparison point. On comparing the average performance of T1 and T2, we observe that the T2 strategy, which exhibits more randomness in the availability of servers performs better. We attribute this to the fact that the average server up-time is larger with T2 than with T1. Hence at any point in time, more servers are available for scheduling.

Impact of placement strategies

In this section, we analyze the impact of different placement strategies. In general, we observe that the number of rejections of policy P1 is always much larger than P2 and P3 since P1 does not take the request popularity into account. In all cases, the random placement policy, P3, has the smallest number of rejections. Though both P2 and P3 take request popularity into account, P3 is better than P2, because the random distribution enlarges the possibility for the replica to be created on more servers

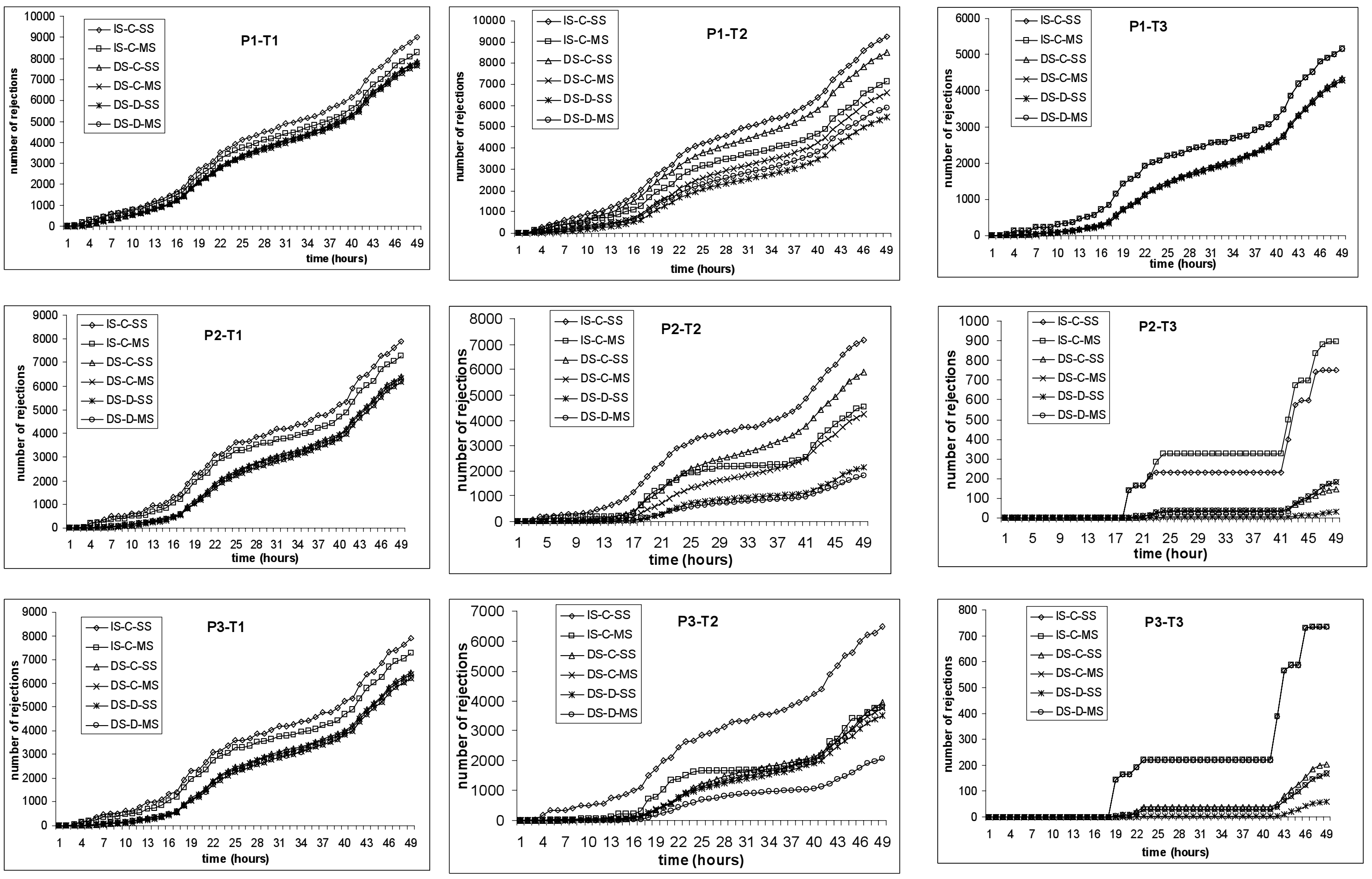

Fig 6. System performance of three time map strategies and three placement strategies - the caption of “Px-Ty” for each figure implies that this result is studied by using the x placement strategies and the y time map model that are explained in Section 5.3. Each figure represents the number of rejections over the time for the six scheduling policies.

with different time maps, so that the servers with the replicas of this video requests can be available for more time.

The Quality-Performance Tradeoff

In this section, we study the tradeoff between the service quality and the system performance. As before, in order to avoid the influence of other unpredictable factors, we choose the P2-T1 configuration. By varying the required network bandwidth of each video from 0.5Mbps to 2Mbps, we get the results shown as Fig. 7. In general, most policies (except IS-C-SS) are sensitive to the quality of the transmission; a reduction in service quality brings fewer rejections. The most dramatic effect is observed in the delayed-start (DS) multiple server (MS) policies, i.e. DS-C-MS and DS-D-MS that can take advantage of resources on any server any time. In the most constrained case, i.e IS-C-SS, a change in resource requirements does not have a significant enough impact on system performance since continuous availability of a single server is of primary concern. In this case, lowering service quality will not help in improving the system performance.

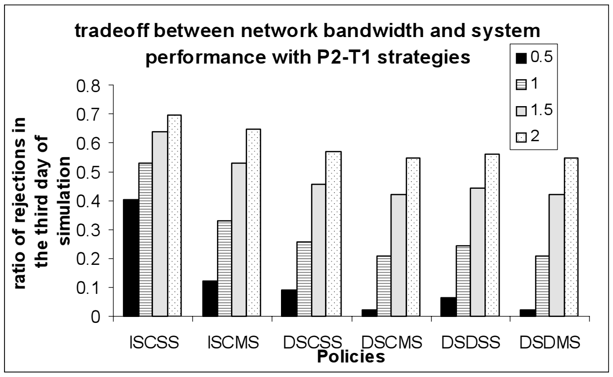

Fig 7. Tradeoff between network bandwidth requirement and system performance - This figure represents the rejection ratio of six scheduling policies for four different network bandwidth requirements.

Other factors affecting the system performance

We also study the impact of server storage and the number of data servers on system performance. We notice that more storage improves the system performance, but only up to a certain point. This is because as storage levels increase, more video objects can be placed causing the rejection rate to fall. But, at higher levels of storage, there is another factor, i.e. network transmission bandwidth, that becomes the prime limiting factor, especially in overload situations. A larger number of servers improves the overall system performance; the flexible policies benefit more from increased number of servers than the constrained policies. With a larger number of servers, more resources are available for multiple server policies. Under the single server policy, e.g. IS-C-SS, system performance cannot improve with a larger number of servers unless the new servers have sufficient continuous service time.

5.5. Summary of performance

In summary, flexibility of resource allocation (as is the case with multiple server delayed start policies) is important to ensure that intermittently available systems perform well. Randomized placement strategies exhibit better performance with more servers available to service a given request at any point in time. While increasing the server storage is useful in decreasing the number of request rejects, this benefit is observed only with low storage levels. At low levels of storage, storage space is the limiting factor; while with high levels of storage, the network bandwidth is the limiting factor. Reducing the quality of service to enhance performance is useful except in the most constrained scheduling environments. Scaling up the system with more servers, i.e. increasing the level of metacomputing, is highly beneficial if multiple server scheduling is applicable.

6. Related work and Future Research Directions

We address related work in existing grid system management and scheduling in computational grid systems. We indicate how the techniques proposed in this paper can be incorporated into existing grid environments.

Legion [9] presents users with wide range of services for security, performance, and functionality. The Resource Allocation Manager (GRAM) of Globus [8] provides the local component for resource management. Resource and computation management services are implemented in a hierarchical fashion. Currently, the Globus Architecture for Reservation and Allocation (GARA) [15] aims to address issues of dynamic discovery, advance or immediate reservation of resources. Our scheduling algorithms can be applied within the *information service* component of the Globus and GARA [17] resource management architectures to support a larger class of applications.

AppLeS [5] performs resource selection as an initial step, and its default scheduling policy chooses the best schedule among the resulting candidates based on the user's performance criteria; other scheduling policies may be provided by the user. Prophet [21], and MARS [4], use only one criterion: the performance goal for all applications is minimal execution time. The adaptive scheduling algorithm proposed in [10] uses a queuing theory based approach for scheduling computationally intensive tasks. NetSolve[7], Nimrod [3] and Ninf [18] are management platforms targeted for scientific applications. The Bricks performance evaluation system introduced in

[19] supports a task model of global computation that can be used for scheduling.

In summary, we have proposed a generalized technique for resource discovery in intermittently available environments and developed specialized policies to deal with varying user requirements. We intend to conduct further performance studies with heterogeneous servers and combinations of request patterns. In order to improve the system scalability and adapt to changing requirements, we are developing dynamic placement strategies for intermittently available environments.

We currently assume a centralized broker that can be a bottleneck as the system scales in size. In order to achieve the scalability of such a system in a large scale heterogeneous network environment, we intend to extend our technique to the hierarchical frameworks [22] for effective and faster wide-area service directory organization and management. Future work will also address resource management in grid based systems with mobile clients. Eventually, we believe that the grid-based infrastructure will play an important role in promoting the widespread use of multimedia applications.

References

[1] M. J. Atallah, Algorithms and Theory of Computation Handbook, page 7-3. 1998.

[2] R.J. Allan. Survey of Computational Grid, Meta-computing and Network Information Tools, Parallel Application Software on High Performance Computers. 1999.

[3] D. Abramson, I.Foster, J. Giddy, A. Lewis, R. Sosic, and R. Sutherst. The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing. In Proc. of the 20th Autralasian Computer Science Conference, Feb. 1997.

[4] M. Buddhikot, G. Parulkar, and J. Cox. Design of a large scale multimedia storage server. In Proc. INET' 94, 1994.

[5] F. Berman and R. Wolski. The AppLeS project: A status report. Proceedings of the 8th NEC Research Symposium, 1997.

[6] A.L.Chervenak. Tertiary Storage: An Evaluation of New Applications, Ph.D. Thesis, UC Berkeley, December, 1994.

[7] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. Tech. Report CS-95-313, 1995.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for Metacomputing system. In The Fourth Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[9] S.J.Chapin, D. Katramatos, J.F. Karpovich, A. Grimshaw, Resource Management in Legion, University of Virginia Technical Report CS-98-09, February 1998.

[10] H. Casanova, M. Kim, J. S. Plank, Adaptive Scheduling for Task Farming with Grid Middleware, 1999.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to Algorithms. MIT, 1999,page 579-599.

[12] A. Dan and D.Sitaram. An online video placement policy based on bandwidth to space ration (bsr). In *SIGMOD* '95, pages 376-385, 1995.

[13] A. Dan, D. Sitaram, P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching, Second Annual ACM Multimedia Conference and Exposition, 1994.

[14] I. Foster, C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure, book, preface, 1998.

[15] I. Foster, A. Roy, V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, International Workshop on Quality of Service, 2000.

[16] Z. Fu and N. Venkatasubramanian. Directory Based Composed Routing and Scheduling Policies for Dynamic Multimedia Environments. Proc. of the IEEE International Parallel and Distributed Processing Symposium 2001.

[17] V. Sander, W. Adamson, I. Foster, A. Roy. End-to-End Provision of Policy Information for Network QoS, 10th IEEE Intl. Symp. on High Performance Distributed Computing, IEEE Press, 115-126, 2001.

[18] S. Sekiguchi, M.Sato, H.Nakada, S. Matsuoka, and U.Nagashima. Ninf: Network Based Information Library for Globally High Performance Computing. In Proc. Of Parallel Object-Oriented Methods and Applications (POOMA), 1996.

[19] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, U. Nagashima, Overview of a Performance Evaluation System for Global Computing Scheduling Algorithm, Proc. Of HPDC 99, pp. 97-104, August 1999.

[20] N. Venkatasubramanian and S. Ramanathan. Load Management in Distributed Video Servers, Proceedings of the Intl. Conference on Distributed Computing Systems, 1997.

[21] J. Weissman and X. Zhao. Runtime support for scheduling parallel applications in heterogeneous NOWS. HPDC 1997.

[22] Dongyan Xu, Klara Nahrstedt, Duangdao Wichadakul, Qos-Aware Discovery of Wide-Area Distributed Services. 2001.