Dynamic RePacking: A Content Replication Policy for Clustered Multimedia Servers [*]

Jonathan Dukes and Jeremy Jones

Computer Architecture Group
Department of Computer Science, Trinity College Dublin

`Jonathan.Dukes@cs.tcd.ie, Jeremy.Jones@cs.tcd.ie`
`http://www.cs.tcd.ie/CAG`

Abstract

Dynamic replication is a technique that can be used by clustered multimedia servers to evaluate the demand for individual streams and selectively replicate or move content to facilitate load balancing. In this paper we explain the motivation for using dynamic replication and provide a brief description of some existing dynamic replication policies. We present a new policy called Dynamic RePacking. This policy is based on the existing MMPacking algorithm [11], but has been modified to handle nodes and files with varying bandwidth and storage characteristics and to reduce the cost of adapting to changes in client demand or server configuration. Simulation results show that Dynamic RePacking performs better than both the original MMPacking algorithm and the threshold-based scheme which we have simulated for comparison. Our results also show that Dynamic RePacking performs significantly less movement and replication of files than the original MMPacking algorithm.

1 Introduction

True on-demand multimedia streaming usually requires that each client has an independent stream that can support interactive operations. Such applications are demanding on both server and network resources. For example, a two hour high quality MPEG-2 movie would require 4Mbps of bandwidth and just over 3.5GB of storage. Using today's commodity PC hardware and software it is possible to supply an on-demand streaming service to a small number of customers. However, the advent of broadband communication in the home means that servers will have to deliver reliable on-demand streaming services cost-effectively to thousands of customers.

The model that has emerged to meet these requirements in other application domains is the *server cluster* [2]. High-availability is achieved because a client request can be handled by more than one node and each node can independently service requests. Scalability is achieved (cost effectively) by adding additional nodes to the cluster. For applications such as web servers and database systems, this model is in widespread commercial use. However, the implementation of large-scale multimedia-on-demand applications in a server cluster environment presents a specific problem: the volume and characteristics of the stored multimedia content means that server performance, scalability, availability and cost are all highly dependent on the location of the content within the cluster.

In this paper, we present a policy for performing partial, selective replication of multimedia content, based on the the changing demand for individual multimedia titles, to facilitate load balancing. We begin in section 2 by describing the motivation for using partial replication and describe some existing

[*]The research described in this paper is part funded by Microsoft Research, Cambridge, UK

replication policies. In section 3, we present our policy, called *Dynamic RePacking*. This policy is based on the existing MMPacking policy [11], which has some useful properties, but must be modified to handle nodes with varying bandwidth and storage capacities and to reduce the cost of adapting to changes in client demand or server configuration. The simulation results presented in section 4 show that our policy performs better than the original MMPacking algorithm, while significantly reducing the number of multimedia files that need to be copied or moved between server nodes to perform load-balancing.

2 Dynamic Replication

Advances in storage technology have made it possible for a single commodity server to supply soft real-time multimedia streams to clients across a network. Multimedia servers based on the single server model, however, exhibit poor scalability and availability [7]. One solution is to “clone” the servers, mirroring available data on each node. This approach increases the bandwidth capacity and availability of the service and is common in web server clusters, where the volume of data stored on the server is small. Cloned servers can be grouped to form a network load-balancing cluster and client requests are distributed among cluster nodes according to their capabilities and current workload. However, the volume of data that is typically stored on a multimedia server usually prohibits this form of complete server replication. For example, to store five hundred typical two-hour DVD quality videos with a bandwidth of 4Mbps would require over 1.5TB of storage at each node. Since most of these videos will only rarely be viewed, cloning multimedia servers in this way is wasteful.

Recent advances in storage area network (SAN) technology have prompted some research into the use of SAN architectures for multimedia servers [6]. One approach is to provide a cluster of front-end streaming nodes with access to shared SAN storage devices, such as disks or RAID storage systems. This solution, however, merely moves scalability and availability problems from the front-end server nodes to the SAN storage devices, since each storage device or RAID storage system has only limited bandwidth capacity for supplying multimedia streams. A solution to the problem of scalability and availability that can be applied to server clusters with locally attached storage may also be applied to SAN architectures. In addition, the high cost of SAN storage means there is still a need for multimedia servers based on server-attached storage.

Server striping is an approach that has received a lot of attention in the past. It is conceptually similar to RAID-0 [10] – multimedia files are divided into equal size blocks, which are distributed among server nodes in a pre-defined order, as illustrated in figure 1. In the context of video-on-demand, servers using this technique are often classified as *parallel video servers* [7]. Server level striping achieves implicit load-balancing across server nodes, while only storing a single copy of each file. Load balancing between server nodes is required to maximise server utilisation and minimise the probability that a client request is blocked [9]. The degree of node interdependence caused by server striping is high, however, because each node only stores part of any file and server nodes need to be used in parallel to supply a single multimedia stream.

Node interdependence has several disadvantages. First, the reconstruction of a single stream from striped blocks can be expensive, particularly if the reconstruction needs to be performed at the server. Secondly, as nodes are added to a parallel multimedia server, the existing content needs to be redistributed to take advantage of the additional storage and bandwidth capacity. Scalability is further hindered by the inability of many server architectures to handle heterogeneous server nodes. For example, Microsoft’s *Tiger Video Fileserver* described in [1] requires that each server node has an identical hardware configuration. Finally, since each file is distributed across all server nodes, the failure of any node will lead to the loss of all streams, unless redundant data is stored [14].

We argue that partial, selective replication of content is a more suitable technique for distributing multimedia content among nodes in clustered multimedia servers. Client demand for individual streams is either continuously or periodically evaluated and this information is used to assign a subset of the available multimedia files to each server node, thereby partitioning the files. Some files may be replicated on multiple nodes to facilitate load-balancing, satisfy client demand or provide fault-

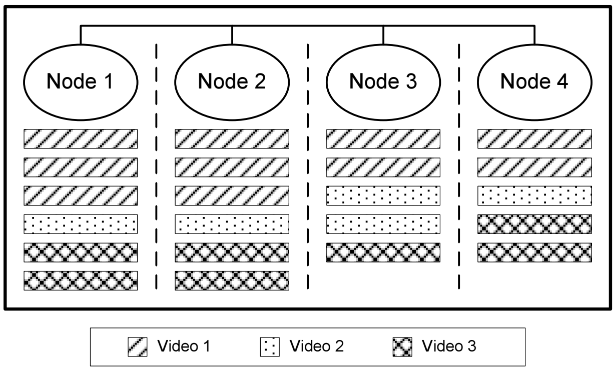

Figure 1: Server striping

tolerance. Since the popularity of each multimedia file can change over time, the assignment of files to nodes must be periodically reevaluated. Since each node can independently supply streams to clients, the degree of node interdependence is minimal. This allows a cluster to be formed from nodes with varying storage and bandwidth capacities, unlike servers based on striping. In addition, nodes can be added to or removed from the server cluster without causing the redistribution of all server content. Although partial replication requires more storage capacity than server striping, the additional cost is predictable and is minimised by replication policies such as the Dynamic RePacking policy described in this paper. The benefit of this additional storage cost is increased scalability and graceful degradation of service when nodes fail.

In the following section, we describe some existing dynamic replication policies.

2.1 Related Work

The *DASD Dancing* scheme [13] performs load-balancing by determining the most suitable node to supply a new stream and reassigning existing streams to different nodes when necessary. Related to this scheme, and of most relevance to our research, is the scheme used to assign multimedia files to nodes. The assignment begins by determining the required number of copies of each file. Files are then assigned to nodes in order to increase the ability of the server to transfer existing streams from one node to another. Finally, static load-balancing is performed, based on the expected demand for individual files, using those files for which only a single copy is required. This does not, however, result in accurate static load-balancing based on the estimated demand for the multimedia files.

The *bandwidth to space ratio (BSR)* policy described in [5] aims to balance the *ratio* of used storage to used bandwidth across server nodes. In contrast, we assign files to nodes to balance expected bandwidth utilisation, and only consider storage capacity as a constraint to this assignment.

A load management policy for multimedia servers, which takes into account server resources other than network and storage bandwidth, is described in [12]. Like our policy, this policy periodically performs predictive placement of files based on estimated demand. The scheme used to allocate files to nodes, however, begins by allocating the most popular file (or the file that will generate the most revenue) first, which may result in the file being stored in only one location. Intuitively, this decreases the ability of the server to perform dynamic load-balancing when assigning new requests to nodes. The ability of this policy to adapt to gradual changes in popularity has not been examined.

All of the above dynamic replication policies assume the existence of data describing the relative demand for each file. In contrast, in this paper, we describe in detail how we estimate the relative demand for each file. The accuracy of this estimation is reflected in our simulation results.

The dynamic segment replication (DSR) scheme [4] divides multimedia files into fixed size segments and performs replication of the segments when the load on a node exceeds some threshold. DSR may be used as a secondary "on-line" replication scheme to allow a server to quickly adapt to unexpectedly high demand for particular files. This use of DSR to complement other replication policies is described in [5]. Such an approach would also complement our Dynamic RePacking policy.

The threshold-based replication policy described in [3] does not attempt to periodically find a suitable assignment of files to nodes. Instead, each time a new stream is started, the server evaluates the unused resources available for supplying streams of the requested file. If this value is below some threshold, an additional replica of the file is created on the node that is currently least loaded. The disadvantage of threshold-based schemes is that expensive replication is delayed until server load is high, reducing the number of requests that can be accepted by the server. The threshold-based replication scheme that we have simulated results in almost full utilisation of any available storage capacity, which may not be desirable. We have simulated this threshold-based policy to compare it with our own Dynamic RePacking policy and our results are presented in section 4.

3 Dynamic RePacking

Dynamic replication policies can be described in three parts: evaluation of the demand for each file, assignment of the files to server nodes and dynamic assignment of client requests to nodes as they arrive.

3.1 Demand evaluation

Usually information about the demand for individual files will not be available from an external source, and needs to be evaluated by the server. When evaluating the demand for a file, several variables need to be considered:

Server load Changes in overall server utilisation may occur on a short-term basis (e.g. between morning and evening). Usually a dynamic replication policy should ignore these fluctuations and only respond to longer-term changes in the relative demand for files, for example, over several days. For this reason, we define the demand, D_i, for a file i as the *proportion* of server bandwidth required to supply the average number of concurrent streams of that file over a given period, which is independent of the actual load on the server. The bandwidth of the server is the sum of the bandwidths of its individual nodes, where a node's bandwidth is the maximum cumulative stream bit-rate that can be supplied by the node.

File popularity The relative popularity of individual multimedia files will change over time, often decreasing as files become older, resulting in changes in the relative demand, D_i, of files. The server should adapt quickly to these changes.

Stream duration The average duration of streams of different files may vary and our definition of demand takes this into account by evaluating the average number of concurrent streams of each file.

Bit-rate The multimedia files stored on a server will usually have different bit-rates, depending for example on the media type (video, audio, etc.) and the level of compression used. Again, our definition of demand takes bit-rate into account.

To evaluate the demand for a file, the Dynamic RePacking policy needs to calculate the average number of concurrent streams of the file, over a period of time of length τ. This value, L_i, for a file i, can be calculated by applying Little's formula [8] as follows:

$$L_i = \lambda_i * W_i, \tag{1}$$

The arrival rate, λ_i, can be calculated by dividing the number of requests for the file, received over a period (R_i), by the length of the period (τ). Similarly, the average stream duration, W_i, can be calculated by dividing the total time spent steaming the file, (Q_i), by the number of requests (R_i). Substituting for λ_i and W_i in equation 1 gives the following formula for evaluating L_i:

$$L_i = \frac{Q_i}{\tau} \tag{2}$$

To calculate D_i for each file, L_i is scaled by the bandwidth, B_i, required to supply a single stream of the file. The following formula can then be used to express the demand for a file as the proportion of server bandwidth required by the file:

$$D_i = \frac{Q_i * B_i}{\sum_{k=0}^{K-1} (Q_k * B_k)} \qquad (3)$$

where K is the number of files stored on the server. Note that the constant measurement period τ cancels in the above equation. Thus, the only measurement required to evaluate the demand for each file is the cumulative duration of all streams of each file, Q_i, over the period τ. The value of D_i for each file can be used at the end of each period as the input to the file placement algorithm. To obtain an average value for D_i over a longer period, a "sliding window" approach is used (Figure 2). The demand for the file is calculated as described above, and the demand for the last T measurement periods is saved. The average demand over the measurement window, $D_i^{'}$ is evaluated as follows:

$$D_i^{'} = \frac{\sum_{t=0}^{T-1} (D_{i,t} * w_t)}{\sum_{t=0}^{T-1} w_t} \qquad (4)$$

where $D_{i,t}$ is the demand for file i during period t and $t = 0$ is the most recent period. Multiplying each term by w_t can be used to give more weight to more recent measurements. Alternatively, values of D_i may be extrapolated to obtain an estimated value for $D_i^{'}$.

3.2 File assignment

In this section, we describe the Dynamic RePacking file assignment algorithm. Our algorithm is a development of the MMPacking algorithm [11] referred to in section 2.1. MMPacking has several attractive properties, which we believe make it a good starting point for the development of a file assignment algorithm for large clustered multimedia servers.

- The packing algorithm causes the most demanding files to have the most replicas. Intuitively, this allows greater flexibility when assigning client requests to nodes to perform load-balancing, since a large proportion of requests will be for files that are assigned to more than one node.
- A combination of high and low demand files are assigned to each node, reducing the impact of node failures and fluctuations from expected demand.
- The algorithm assigns at most $\frac{K}{N} + 1$ files to each node. Assuming the average size of the files stored on a node is the same for all nodes, it is possible to estimate the minimum storage capacity required by MMPacking.

MMPacking, however, needs to be modified before we can use it in a clustered multimedia server. We begin by providing an overview of the original MMPacking algorithm, but instead of using the popularity of files to determine their placement, we use the demand, D_i, as described above. This modification allows the server to handle files with different bit-rates and different average stream durations. Consider a distributed multimedia server with N nodes, $S_0, \cdots, S_{N-1}$, which stores K multimedia files, $M_0, \cdots, M_{K-1}$. Each file M_i has demand D_i. It is assumed that $K \gg N$ and that each node has the same bandwidth. Initially, the files are sorted in ascending order of demand. Files are removed from the beginning of the *file list* and assigned to server nodes in a round-robin fashion, as illustrated in Figure 3(a). The cumulative demand for a node is the sum of the demand for each file assigned to that node. If after assigning a file to a node, the cumulative demand of the node exceeds $1/N$, then the demand for the last file placed on the node is adjusted so the cumulative demand of the node is equal to $1/N$. A new *replica file* with the shortfall in demand is added to the beginning of the file list and the algorithm proceeds to the next node. Finally, if after assigning a *replica* file to a node the cumulative demand for the node is less than $1/N$, then the algorithm continues by assigning the next file to the *same* node (Figure 3(b)). A final assignment of files to nodes is illustrated in Figure 3(c). When all files have been assigned, the cumulative demand of each node will be $1/N$.

We now describe our Dynamic RePacking packing algorithm as a series of developments to MMPacking. The first development allows the algorithm to handle nodes with varying bandwidth capacities and is based on the following property of the MMPacking algorithm. After assigning a file to a node, assignment continues with the next node, which will always have either the same or smaller cumulative demand as the current node (after assigning the file). This was shown to be true for

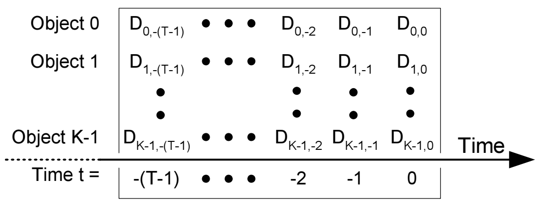
Object 0
Object 1
Object K-1
Time t =
Time
$D_{0,-(T-1)}$ $D_{0,-2}$ $D_{0,-1}$ $D_{0,0}$
$D_{1,-(T-1)}$ $D_{1,-2}$ $D_{1,-1}$ $D_{1,0}$
$D_{K-1,-(T-1)}$ $D_{K-1,-2}$ $D_{K-1,-1}$ $D_{K-1,0}$
-(T-1) -2 -1 0

Figure 2: Sliding window of data for estimating D_i.

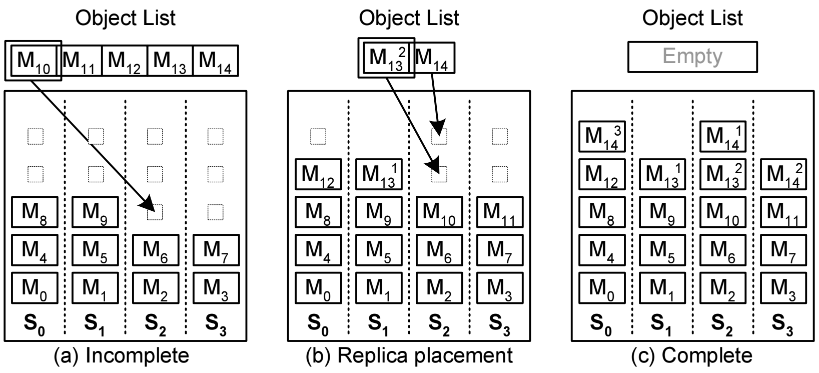

Figure 3: Illustration of original MMPacking algorithm with four nodes and 14 files. Files M_{13} and M_{14} have been replicated.

MMPacking by observing that the cumulative demand for nodes, after round l of file placements, has the following property:

$$C_0^l \leq C_1^l \leq \cdots \leq C_{N-1}^l \leq C_0^{l+1} \leq \cdots \tag{5}$$

where C_n^l is the cumulative demand for node n after a file has been assigned during round l. (As described above, the cumulative demand of a node is the sum of the demand for each file assigned to the node.) This property holds true because files are assigned to nodes in ascending order of demand. As described above, a desirable property of MMPacking is that replication only occurs for the most popular files. If the original MMPacking algorithm was applied to nodes with different bandwidths, replication may be forced for files that could still be placed, without replication, on other nodes. Conversely, popular files that would usually be replicated many times may be replicated less. To avoid this our algorithm assigns files in the following manner. Initially, we calculate for each node, j, a target cumulative demand, G_j, which represents the bandwidth of the node as a proportion of the total server bandwidth:

$$G_j = \frac{B_j}{\sum_{n=0}^{N-1} B_n} \tag{6}$$

where B_j is the bandwidth of node j. We define the *target shortfall*, H_j^l, of a node j as the difference between the cumulative demand of the node – after a file has been assigned to the node during round l – and the target cumulative demand of the node:

$$H_j^l = G_i - C_j^l \tag{7}$$

Server nodes are sorted in *descending* order of target shortfall, H_j^l, thus maintaining the property described by equation 5. Assignment begins in the same manner as MMPacking. If after assigning a file to a node the target shortfall is still greater than the shortfall for the next node (from the previous round of assignments), then the current round of assignments ends prematurely and the algorithm begins a new round, starting with the first node. This is illustrated in Figures 4(b) to (d), where files are repeatedly assigned to S_0 until the shortfall of the node is less than the shortfall on S_1. Formally, the current round l of assignments ends with node j if either of the following conditions holds true:

$$H_j^l > H_{j+1}^{l-1}, \qquad 0 \leq j \leq N-2$$

or

$$j = N - 1 \tag{8}$$

Otherwise, file assignment continues as for MMPacking. When every file has been assigned, the target shortfall of each node will be zero. This algorithm produces the same result as MMPacking if every node has the same bandwidth. Otherwise, if nodes have different bandwidths, load-balancing is still performed but nodes with higher bandwidths will store more files.

The main aim of the Dynamic RePacking algorithm is to reduce the cost of adapting the server to changes in the demand for individual files, the number of nodes in the server or the number of files to be stored. Each invocation of the original MMPacking algorithm is independent of the previous one, which could potentially result in every file being moved from one node to another. The cost of this reorganisation makes MMPacking impractical in many environments. To reduce this cost, Dynamic RePacking attempts to re-pack files where they were previously stored. Previously, the file with the lowest demand was assigned to a node. This time, however, the file with the lowest demand *currently stored on the node* is reassigned to the node. If no suitable file exists, the file with the lowest demand is selected as before. To allow the algorithm to pack nodes with the most bandwidth first, the nodes need to be resorted by decreasing target shortfall, G_j^l, every time we start round $l+1$ of allocations (Figure 4(f)). The algorithm then proceeds as before.

Pseudo-code for the algorithm, as described so far, is shown in Figure 5. The `SelectFile(...)` and `NextNode(...)` procedures select the object to be packed and the next node to proceed with, as described above.

It may be desirable to create at least one replica of some or all files for fault-tolerance, and we briefly describe how this may be done. Packing is performed in two phases. An initial *test pack* determines the unallocated storage capacity after packing without additional replicas. Files are assigned additional replicas in decreasing order of demand until each file reaches the desired minimum number of replicas or until all unallocated storage capacity has been used. The Dynamic RePacking algorithm is again used to determine the final assignment of files to nodes, creating the desired number of additional replicas for each file. If, when assigning a file to a node, the demand for the file will be entirely satisfied but the file has not reached its required minimum number of replicas, then only $\frac{1}{P}$ of the remaining unsatisfied demand for the file is assigned to the current node, where P is the number of replicas that are still to be created.

Like the original MMPacking algorithm, the algorithm described above can be "blocked" on a particular node if there is insufficient storage capacity to store the assigned file. Dynamic RePacking handles this situation by packing the most demanding files at the expense of the least demanding ones, thus reducing the imbalance between storage and bandwidth utilisation. If at any stage during the execution of the algorithm there is insufficient storage capacity to assign a file to a node, we repeatedly remove the least demanding file from the node until sufficient storage capacity is available. The removed files are added to a *victim list* and the algorithm continues. If the combined demand for the files that need to be removed from a node is greater than or equal to the demand for the file being placed, then the removal is abandoned and the procedure is repeated for the next file in the file list, which will have a higher demand. If no file can be placed, we proceed to the next node. The files on the victim list will have relatively low demand and can be assigned to nodes later, without causing any significant load imbalance.

3.3 Service assignment

As client requests arrive, they are simply assigned to the least-utilised node with a copy of the requested file. An enhancement of the policy might attempt to move existing streams from one node to another, in a similar way to the DASD Dancing policy [13].

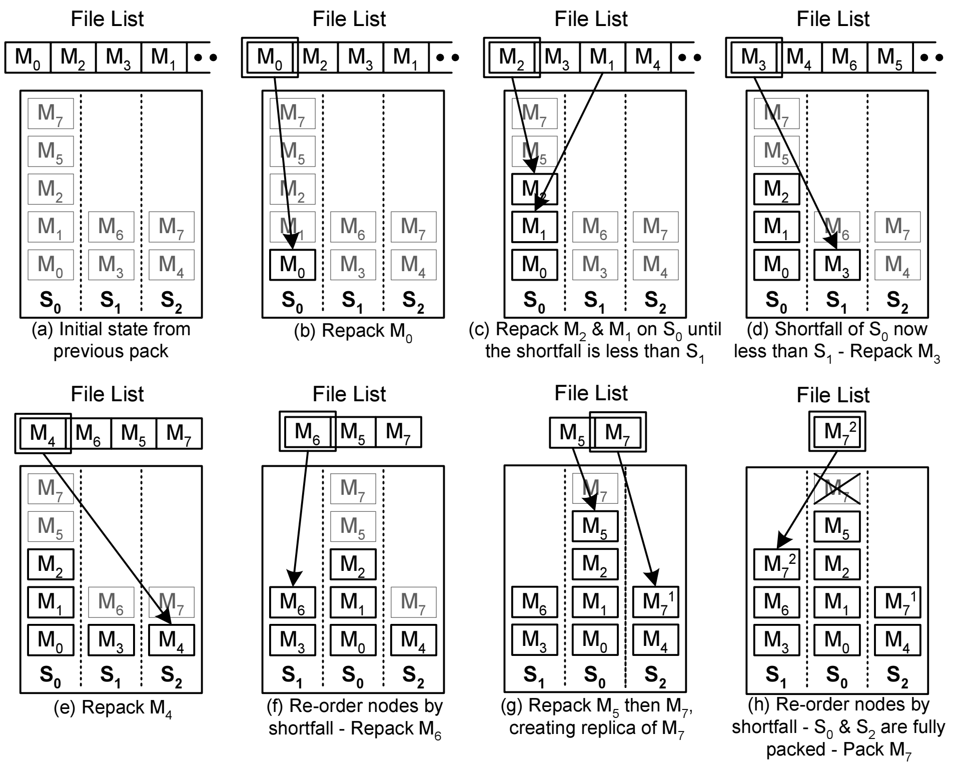
File List
(a) Initial state from previous pack
(b) Repack M0
(c) Repack M2 & M1 on S0 until the shortfall is less than S1
(d) Shortfall of S0 now less than S1 - Repack M3
(e) Repack M4
(f) Re-order nodes by shortfall - Repack M6
(g) Repack M5 then M7, creating replica of M7
(h) Re-order nodes by shortfall - S0 & S2 are fully packed - Pack M7
S0
S1
S2

Figure 4: Illustration of the basic Dynamic RePacking Algorithm with three nodes and eight files to be assigned. The initial assignment was based on an ordering of files $(M_0, M_1, M_2, \ldots, M_7)$ which has been re-ordered due to a change in the popularity distribution. After this assignment, the only file that needs to be moved is M_7, from S_0 to S_1.

```
files.SortUp();
ndIdx = 0;
while(files.Count > 0) {
  if(ndIdx == 0)
    nodes.SortDown();

  if(nodes[ndIdx].TargetShortfall > 0)
  {
    fileIdx = SelectFile(ndIdx);
    allocation = Min(nodes[ndIdx].TargetShortfall, files[fileIdx].Demand);
    PackFile(ndIdx, fileIdx);
    nodes[ndIdx].TargetShortfall -= allocation;
    files[fileIdx].Demand -= allocation;

    if(!files[fileIdx].IsReplica || nodes[ndIdx].TargetShortfall == 0)
      ndIdx = NextNode(ndIdx);

    if(files[fileIdx].Demand == 0)
      files.Remove(objIdx)
    else
      files.SortUp();
  }
  else
    ndIdx = NextNode(ndIdx);
}
```

Figure 5: Basic Dynamic RePacking algorithm

4 Simulation

We have developed a discrete event simulation to evaluate the performance of the Dynamic RePacking policy for various server configurations. We have also simulated the MMPacking algorithm, upon which our policy is based, and the threshold-based replication policy described in [3] for comparison. A framework was developed for the simulation of different replication policies. This framework defines those server characteristics that are common to each server simulation. For example, the allocation of server resources to client streams and file replication streams, the generation of client requests and the mechanism for adding or removing copies of a file is the same for the three policies that we have simulated. Thus, each simulation differs only in the way the demand for each file is estimated and in the assignment of files to nodes.

Request inter-arrival times are generated from an exponential distribution whose mean varies on a periodic basis, representing variations in overall server load (for example, between morning and evening) between 0.4 (lightly loaded) and 1.0 (fully loaded). Individual objects are requested with a frequency determined by a Zipf distribution with the parameter $\theta = 0$, as used by Chou et al[3], so object i is requested with probability $P(i)$:

$$P(i) = \frac{1}{i\left(\sum_{j=1}^{K}\frac{1}{j}\right)} \tag{9}$$

where K is the number of multimedia objects available on the server. In addition, the popularities of the objects are "shuffled" periodically, again using the same technique as Chou et al. to simulate a gradual shifting of popularity between objects. The probability of requesting object i during period t' is based on the probability distribution during the previous period t, as follows, assuming K is even:

$$p_i(t') = \begin{cases} p_{i+2}(t) & \text{if } i \text{ is even and } 0 \le i < K-2 \\ p_K(t) & \text{if } i = K-2 \\ p_{i-2}(t) & \text{if } i \text{ is odd and } 1 < i \le K-1 \\ p_1(t) & \text{if } i = 1 \end{cases} \tag{10}$$

We have simulated two types of server configuration, one with identical nodes supplying streams of files with identical bit-rates, sizes and playback times (homogeneous configuration), and another with different nodes and files (heterogeneous configuration).

4.1 Homogeneous configuration

To evaluate the scalability of each of the simulated policies, we have simulated server configurations with numbers of nodes ranging from 4 to 32. Two scale-out scenarios were examined. In the first scenario, the cumulative server bandwidth was increased while keeping the cumulative storage capacity of the server constant. For example, in a server configuration with 8 nodes, each node had the same bandwidth but half the storage capacity as those in a server with only 4 nodes. In the second scale-out scenario, both the cumulative bandwidth and storage capacities of the server were increased. For example, the nodes in a configuration with 8 nodes were identical to those in a configuration with only 4 nodes. In this second scenario, the number of files stored was also increased in proportion to the cumulative storage capacity of the server. In each scenario, we measured the proportion of client requests that were accepted immediately by the server, the average number of files that needed to be moved or replicated between successive "shuffles" of file popularity and the proportion of available server storage capacity used by the replication policy. The simulation parameters are shown in Table 1 and the results are summarised in Figures 6 and 7.

In both scenarios, and in contrast with the other policies, the acceptance rate of the Dynamic RePacking policy is consistently over 98% for all simulated server configurations. One of the main aims of the Dynamic RePacking policy is to reduce the cost of adapting the server to changes in demand for individual files, thereby decreasing the workload on server nodes. Our results show that the rate of replication and movement of files for Dynamic RePacking is significantly lower than that for MMPacking. Although threshold based replication achieves an even lower rate of replication and movement, this is due to the full storage capacity utilisation that results from this policy, preventing replicas from being created in a timely fashion. In contrast, the Dynamic RePacking policy uses between 84% and 91% of the available storage in scale-out scenario 1 and between 86% and 88% in scenario 2. In both cases, the minimum proportion of storage capacity required to store a single copy of each file is 83.3%. This is the same as the storage cost for a server based on striping (without redundancy) rather than replication. Our results show that, for server configurations with up to 32 nodes, Dynamic RePacking uses less than 9% more storage than a server based on striping. (Using server-attached storage technology, this represents an additional cost of less than €1000 per terabyte at 2002 prices.) In most cases, MMPacking uses a higher proportion of server storage capacity due to excessive movement of files, which much exist in two locations while they are moved. In scenario 1 with four nodes, however, there is insufficient bandwidth to move files to their assigned nodes, leading to under-utilisation of storage capacity and a low client acceptance rate.

It has been shown [11] that the original MMPacking algorithm will replicate no more than $N-1$ files, where N is the number of nodes in the server, with no more than one replica on each node. If all nodes in a server have identical storage capacities, the minimum storage requirement for the server is $K+N$ files. This does not, however, take into account the additional space required to allow files to be moved from one node to another. Our results suggest that $K+2N$ is a more realistic minimum storage capacity for heterogeneous servers using Dynamic RePacking and our results also show that the average storage utilisation for Dynamic RePacking does not exceed this value.

In summary, for heterogenous configurations, Dynamic RePacking achieves a higher acceptance rate, while using less storage than the other simulated policies.

4.2 Heterogeneous configuration

We have repeated the second experiment described above (scaling both bandwidth and storage capacity) for heterogeneous server configurations. Each node had a bandwidth chosen at random and a storage capacity that was proportional to its bandwidth and each file had a bit-rate and size that were also chosen at random. The playback time for each file was proportional to its size and inversely proportional to its bandwidth. The parameters are shown in Table 2 and the results are summarised in Figure 8. (The MMPacking algorithm was not simulated since is cannot handle heterogeneous server configurations. Addressing this limitation was one of the goals of the Dynamic RePacking policy.)

The results show that the Dynamic RePack-

	Scale-out Scenario 1	Scale-out Scenario 2
Number of nodes (N)	$4, 8, 12, \ldots, 32$	$4, 8, 12, \ldots, 32$
Number of files (K)	400	$25N$
File size (units)	10	10
File bit-rate (units)	1	1
Node storage (units)	$4800/N$	300
Node bandwidth (units)	160	160
Evaluation period	τ (Dynamic RePacking) 8τ (MMPacking)	τ (Dynamic RePacking) 8τ (MMPacking)
Number of periods	8 (Dynamic RePacking) 1 (MMPacking)	8 (Dynamic RePacking) 1 (MMPacking)
Stream duration	$\tau/2$	$\tau/2$
Simulation time	600τ	600τ
Shuffle period	24τ	24τ
Server load	$0.4 \ldots 1.0$	$0.4 \ldots 1.0$

Table 1: Parameters for simulation of homogeneous server configuration.

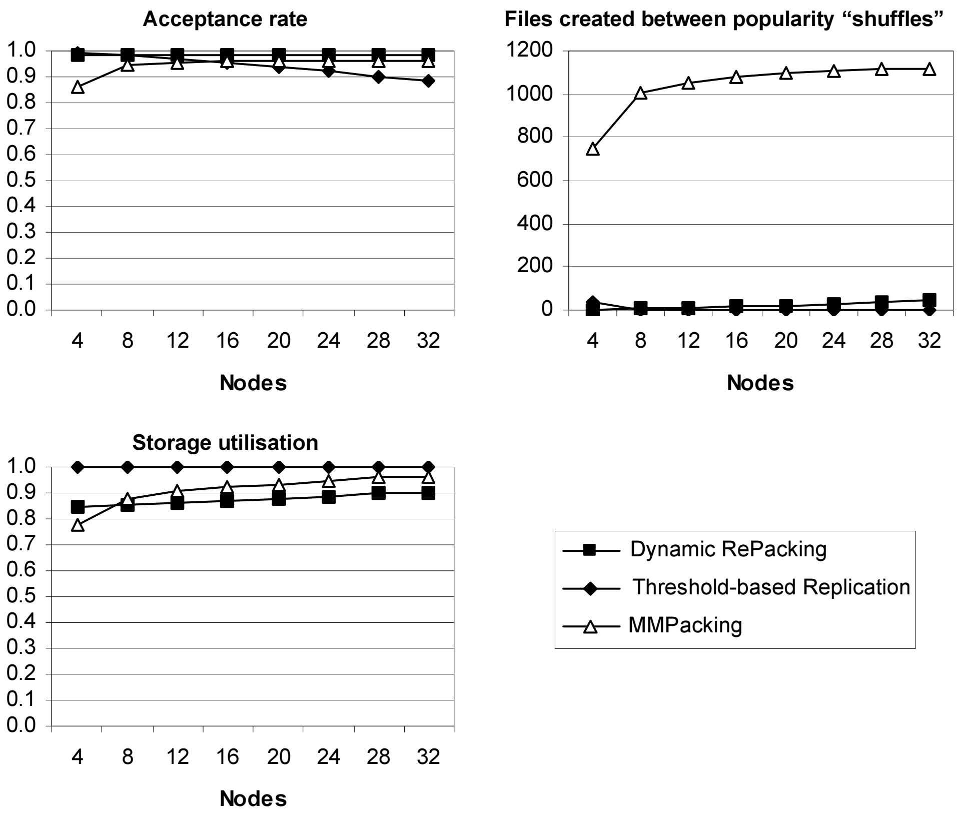

Figure 6: Scale-out Scenario 1 – Adding nodes to increase the cumulative server bandwidth while maintaining a constant cumulative server storage capacity

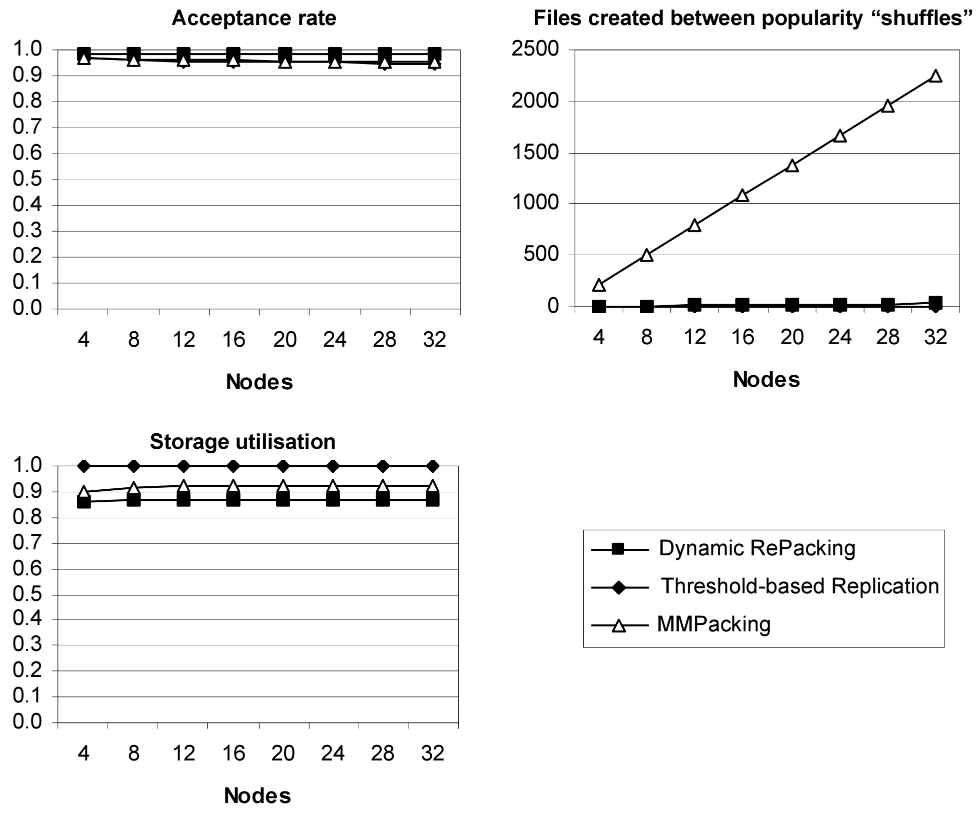

Figure 7: Scale-out Scenario 2 – Adding nodes to increase both the cumulative server bandwidth and cumulative server storage capacity, while also increasing the number of files stored

Number of nodes (N)	$4, 8, 12, \ldots, 32$
Number of files (K)	$25N$
File size (units)	$1 \ldots 15$
File bit-rate (units)	$1 \ldots 5$
Node storage (units)	$160 \ldots 320$
Node bandwidth (units)	100
Evaluation period	τ
Number of periods	8
Stream duration	$\frac{\tau \times size}{15 \times bit-rate}$
Simulation time	600τ
Shuffle period	24τ
Server load	$0.4 \ldots 1.0$

Table 2: Parameters for simulation of heterogeneous server configuration.

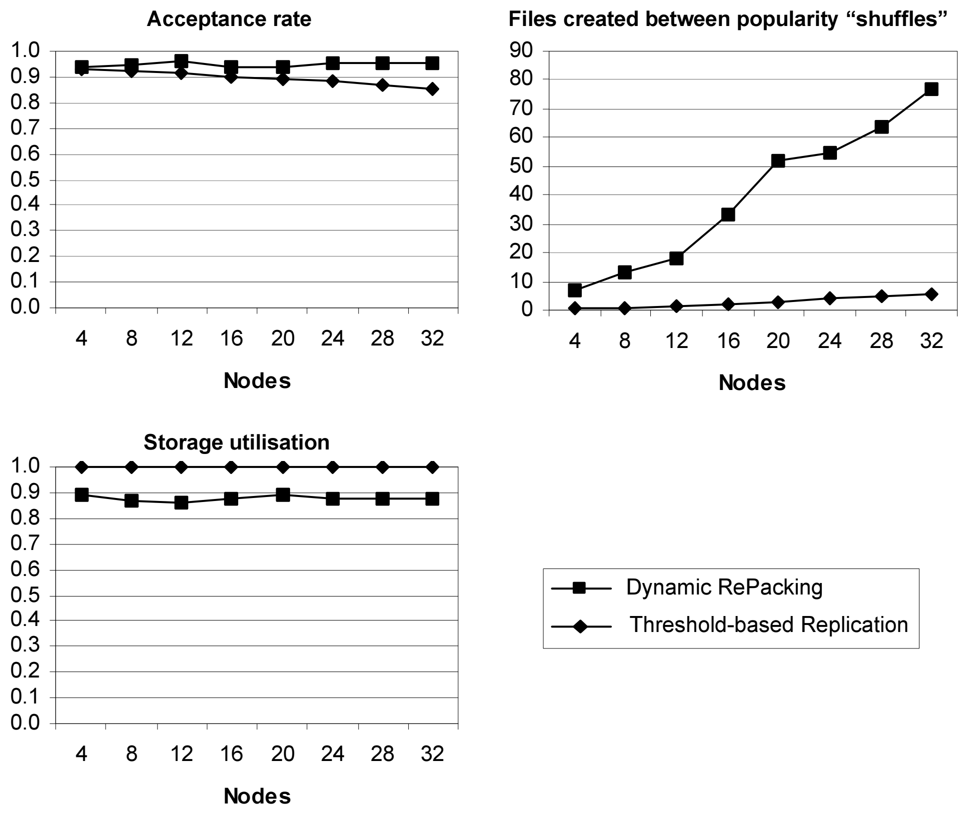

Figure 8: Simulation of heterogeneous server configurations

ing algorithm achieves a higher acceptance rate than threshold-based replication for heterogeneous server configurations, at the expense of a higher rate of file movement. As for homogeneous server configurations, however, the low rate of file movement for threshold-based replication may be explained by the high level of storage utilisation (100% compared with 88.7% for the Dynamic RePacking policy) which restricts replication and reduces the acceptance rate. For a configuration with 32 nodes Dynamic RePacking creates almost 80 new replicas every shuffle period. Given the long period between successive popularity shuffles, however, this frequency of replication is acceptable. The acceptance rates for heterogeneous server configurations are also slightly lower than those for the equivalent homogeneous configurations shown in Figure 7 and the rate of file movement and replication and the storage utilisation are slightly higher, indicating the cost of deviating from homogeneous server configurations.

5 Conclusions and Future Work

In this paper, we began by explaining our motivation for using dynamic replication of content to provide scalability and availability in clustered multimedia servers. Although there have been recent advances in storage technology, we believe that, regardless of the storage architecture used, selective replication of content based on client demand is still required to allow clustered multimedia servers to provide terabytes of content to thousands of clients.

We have described in detail the *Dynamic RePacking* policy. The policy is based on the MMPacking algorithm [11], but has been modified to handle nodes with varying storage and bandwidth capacities and titles with varying bit-rates and sizes, and to reduce the cost of adapting to changes in client behaviour. We have developed a discrete event simulation to compare the performance of Dynamic RePacking with that of MMPacking and a threshold-based replication policy. Our results indicate that Dynamic RePacking achieves a higher client acceptance rate than either of the other two replication policies, while utilising a lower proportion of the available storage capacity.

A prototype clustered multimedia server that uses the Dynamic RePacking policy is currently under development. The results of the work described in this paper indicate that policy will allow the server to scale easily, while requiring little user intervention.

References

[1] W. Bolosky, J. Barrera, R. Draves, R. Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, and R. Rashid. The tiger video fileserver. In *Proceedings of NOSSDAV'96*, April 1996.

[2] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, July/August 2001.

[3] C. Chou, L. Golubchik, and J. Lui. Striping doesn't scale: How to achieve scalability for continuous media servers with replication. In *Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS)*, pages 64–71, April 2000.

[4] A. Dan and D. Sitaram. Dynamic policy of segment replication for load-balancing in video-on-demand servers. *ACM Multimedia Systems*, 3(3):93–103, July 1995.

[5] A. Dan and D. Sitaram. An online video placement policy based on bandwith to space ratio (BSR). In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 376–385. ACM Press, May 1995.

[6] A. Guha. The evolution to network storage architectures for multimedia applications. In *Proceedings of the IEEE Conference on Multimedia Computing Systems*, pages 68–73, June 1999.

[7] J. Y. B. Lee. Parallel video servers: A tutorial. *IEEE Multimedia*, 5:20–28, April–June 1998.

[8] J. D. C. Little. A proof of the queuing formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, 1961.

[9] T. D. C. Little and D. Venkatesh. Popularity-based assignment of movies to storage devices in a video-on-demand system. *ACM Multimedia Systems*, 2(6):280–287, January 1995.

[10] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109–116, 1988.

[11] D. N. Serpanos, L. Georgiadis, and T. Bouloutas. MMPacking: A load and storage balancing algorithm for distributed multimedia servers. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(1):13–17, February 1998.

[12] N. Venkatasubramanian and S. Rananathan. Load management in distributed video servers. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS '97)*, May 1997.

[13] J. L. Wolf, P. S. Yu, and H. Shachnai. DASD dancing: A disk load balancing optimization scheme for video-on-demand computer systems. In *Proceedings of ACM SIGMETRICS '95*, pages 157–166, May 1995.

[14] P. C. Wong and Y. B. Lee. Redundant arrays of inexpensive servers (RAIS) for on-demand multimedia services. In *Proceedings ICC 97*, pages 787–792, June 1997.