# DYNAMIC REPLICATION OF CONTENT IN THE HAMMERHEAD MULTIMEDIA SERVER

Jonathan Dukes
Jeremy Jones
Department of Computer Science
Trinity College Dublin, Ireland
Email: Jonathan.Dukes@cs.tcd.ie

**KEYWORDS**

Multimedia servers, server clusters, video-on-demand, group communication, replication.

**ABSTRACT**

In a clustered multimedia server, by periodically evaluating client demand for each file and performing selective replication of those files with the highest demand, the files can be distributed among server nodes to achieve load balancing. This technique is referred to as *dynamic replication*. Several dynamic replication policies have been proposed in the past, but to our knowledge, our work is the first to describe in detail the implementation of dynamic replication in a server cluster environment. In this paper, we describe the architecture of the HammerHead multimedia server cluster. HammerHead has been developed as a cluster-aware layer that can exist on top of existing commodity multimedia servers – our prototype takes the form of a plug-in for the multimedia server in Microsoft Windows Server 2003™. Replicated state information is maintained using the Ensemble group communication toolkit. We briefly describe our own dynamic replication policy, Dynamic RePacking, and its implementation in the HammerHead server. Finally, we present early performance results from a prototype version of the HammerHead server.

**INTRODUCTION**

The server cluster model is widely used in the implementation of web and database servers. High-availability and scalability are achieved by combining commodity hardware and software, server clustering techniques, storage technologies such as RAID and storage area networks (SANs). However, the implementation of large-scale on-demand multimedia servers in a cluster environment presents specific problems.

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. Such servers, however, exhibit poor scalability and availability (Lee, 1998). 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. In addition, since the majority of multimedia files are rarely requested by clients, replication of low-demand files is wasteful. (We use this model as a performance baseline for our server.)

The use of storage area network (SAN) technology in multimedia servers has also been investigated in the past (Guha, 1999). 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. Scalability and availability problems are, however, merely moved from front-end nodes to SAN storage devices, since the aggregate server bandwidth is likely to exceed the bandwidth of the SAN storage devices. In this case, any solution that can be applied to servers with directly attached storage devices may equally be applied to SAN architectures.

*Server striping* has been used in the past to share workload between multimedia servers. The concept is similar to RAID-0 (Patterson et al., 1988) – multimedia files are divided into equal size blocks, which are distributed among server nodes in a predefined order (Lee, 1998). Implicit load-balancing across server nodes is achieved, while only storing a single copy of each file. The degree of node interdependence caused by server striping is high, however, because each server node is used in parallel to supply each individual multimedia stream. Node interdependence has several disadvantages. First, the process of stream reconstruction is expensive. Secondly, as nodes are added to a server, existing content must be redistributed. Many architectures also require that each sever node has an identical hardware configuration (Bolosky et al., 1997). Finally, the failure of any single node will lead to the loss of all streams, unless redundant data is stored (Wong and Lee, 1997).

We argue that partial, selective replication of content is a more appropriate technique for distributing content among nodes in a clustered multimedia server. Client demand for individual multimedia files is periodically evaluated and this information is used to allocate a subset of the files to each node. The subsets are chosen to approximate an even distribution of server workload across all server nodes. Some files may be replicated to satisfy client demand or facilitate load balancing. The popularity of individual files will change over time, so the assignment of files to nodes must be reevaluated periodically.

Since each node can independently supply streams of the files it stores to clients, the degree of node interdependence is minimal. Thus, servers can be constructed from nodes with varying

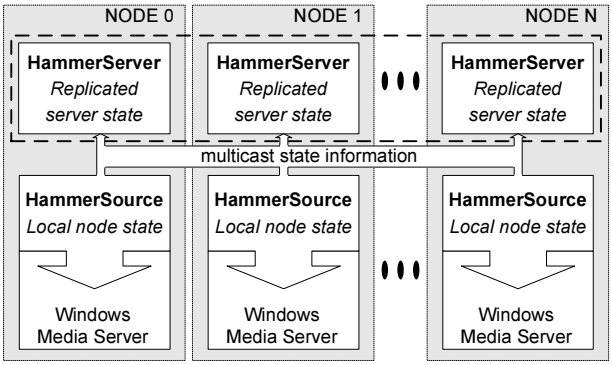


Figure 1: HammerHead Architecture

storage and bandwidth capacities and nodes an be added or removed without redistributing all existing content. Although partial replication may require more storage capacity than server striping, the additional cost is predictable and is minimised by replication policies such as the Dynamic RePacking policy implemented by our HammerHead multimedia server.

Although past research has produced other dynamic replication policies, we believe our work is the first to examine in detail the implementation of a dynamic replication policy in a cluster environment. This paper describes the architecture of the HammerHead clustered multimedia server. Rather than develop our own multimedia streaming software, Hammerhead has been designed as a cluster-aware layer that integrates with existing multimedia server implementations. Our prototype server uses a plug-in to integrate with the multimedia server in Microsoft Windows Server 2003™, although it could easily be adapted to integrate with other commodity multimedia servers. We have used the Ensemble group communication toolkit to provide reliable inter-node communication, facilitate maintenance of replicated server state and detect node failure.

In the next section, we describe the architecture of the HammerHead clustered multimedia server. We then briefly describe the implementation of our content replication policy in the context of the HammerHead server and we present performance results. We will also briefly discuss other related work.

## HAMMERHEAD ARCHITECTURE

A HammerHead server cluster consists of three main components (illustrated in Figure 1).

### Multimedia Server

Rather than develop our own multimedia streaming server, we have designed HammerHead as a cluster-aware layer on top of an existing commodity multimedia server. For our prototype server, we have chosen to use the multimedia server in Microsoft Windows Server 2003™.

### HammerSource

This component was developed as a plug-in for the multimedia server described above, but could easily be adapted for use with other multimedia servers. It captures the state of the multimedia server (source) on a single node and publishes the state in the HammerServer layer, described below. The captured state includes the server's capabilities, properties of any stored multimedia content and the state of any client streams being supplied by the server. Each HammerSource functions independently of HammerSources on remote nodes.

### HammerServer

Each HammerServer instance maintains a replica of the combined state information published by the HammerSources in the group. Client requests are initially directed to any available HammerServer, which uses the replicated server state to redirect the client request to the multimedia server on a suitable node. The choice of node will depend on the load balancing policies implemented by the server. The HammerServer is also responsible for implementing the Dynamic RePacking content placement policy described later in the paper.

It is intended that the HammerServer component will eventually present an image of a single virtual multimedia server to both streaming clients and to other geographically distributed multimedia servers. This would allow, for example, a HammerHead cluster to participate in a content distribution network (CDN).

Although Figure 1 shows a HammerServer instance on each node, this is not a requirement. Only one HammerServer instance must exist to perform client redirection and manage the HammerSources present in the cluster. Any additional HammerServers will increase the availability of the client redirection service and provide additional replicas of the global state.

### Cluster Communication

We have used the Ensemble group communication toolkit (Hayden and van Renesse, 1997) to provide reliable unicast and multicast communication between HammerSources and HammerServers. Ensemble provides application programmers with mechanisms for performing reliable message-based communication between a group of processes and detecting the failure of processes in the group. Specifically, we have used the Maestro open toolkit (Vaysburd, 1998), a set of tools that runs on top of Ensemble to to allow application programmers to work with object-oriented group communication abstractions.

An implementation model facilitated by Maestro that corresponds to the HammerHead architecture described above is the *client/server model with state transfer*. In this model, group members are designated as either *server members* or *client members*. Only server members participate in the state transfer protocol. Both clients and servers can reliably send messages to any single group member, to a subset of members, to the set of all servers in the group or to the entire group. HammerServers are *server* group members and HammerSources are *client* group members.

When a HammerSource joins a group containing one or more HammerServers, it multicasts its current state information to the HammerServers in the group. Each HammerServer will then

merge the HammerSource's state into the global server state. When an event (for example, the start of a stream or the addition of a new file) causes a change in the state of a HammerSource, a message describing the event is multicast to the set of HammerServers, which will update the global state accordingly.

Similarly, if after executing the Dynamic RePacking content placement policy, a HammerServer determines that a file must be copied or removed, it uses Ensemble to send a message to the corresponding HammerSource, which is then responsible for making the required changes.

When a HammerServer joins a group already containing one or more HammerServers, the Maestro state transfer protocol is initiated. During the execution of the state transfer protocol, Maestro will prompt one of the existing HammerServers to send its state to the new server. When this state information has been received and stored, the new server informs Maestro that the state transfer is complete. Any messages sent during the state transfer, which are not related to the state transfer itself, will be delayed and sent when the transfer is complete. Thus, when a new HammerServer joins the group, every HammerServer, including the new one, will contain the same server state and will receive exactly the same set of messages, resulting in a consistent replicated state.

It is worth noting that to maintain a reasonable level of performance and avoid locking the server state for long periods, HammerHead uses an asynchronous communication model. For example, when a HammerServer sends a request to a HammerSource to obtain a copy of a file, it does not wait for a response. Instead, the server will only see the result of the request when the source informs it of the presence of the new file.

In the following section, we describe how the HammerHead server cluster implements the Dynamic RePacking content placement policy.

## DYNAMIC REPACKING

The HammerServer component performs three related functions: evaluation of client demand for each file on the server, assignment of a subset of those files to each node and redirection of client requests to suitable nodes for streaming.

### 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 HammerServer. 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.

**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 HammerServer needs to calculate the average number of concurrent streams of the file, over a period of time of length $\tau$. This value, $L_i$, for a file $i$, can be calculated by applying Little's formula (Little, 1961) ($L_i = \lambda_i.W_i$). The arrival rate, $\lambda_i$, can be calculated by dividing the number of requests for the file, received over a period, by the length of the period, $\tau$. Similarly, the average stream duration, $W_i$, can be calculated by dividing the cumulative duration of all streams of the file, $Q_i$, by the number of requests. Substituting for $\lambda_i$ and $W_i$ in Little's formula, $L_i$ may be evaluated as follows:

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

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 expected number of concurrent streams of the file:

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

where $K$ is the number of files stored on the server. Thus, the only measurement required by a HammerServer to evaluate the demand for each file is the cumulative duration of all streams of each file, $Q_i$, over the period $\tau$. The value of $D_i$ for each file can be used at the end of each period as the input to the file assignment algorithm. To obtain an average value over a longer period, a "sliding window" approach is used, where the demand for the last $T$ measurement periods is saved. The average demand for a file $i$, $D'_i$, is estimated by taking a weighted average of all of the measurements of demand in the sliding window.

Each time a stream of a file begins on a node, the HammerSource on that node will multicast information about the new stream to the set of HammerServers. Similarly, when the stream ends, the HammerSource will multicast an end-of-stream message to each HammerServer. When a stream ends, each HammerServer will calculate the duration of the stream and add it to the cumulative stream duration for the corresponding file. At

the end of each period, $\tau$, each HammerServer uses the cumulative stream duration of each file to estimate the demand using the technique described above. This information is then used to assign files to nodes, replicating files if necessary, as described in the next section.

**File Assignment**

In this section, we give a brief overview of the Dynamic RePacking file assignment algorithm. A more comprehensive description can be found in our prior work on Dynamic RePacking (Dukes and Jones, 2002). Our policy is based on a number of modifications to the MMPacking policy described in (Serpanos et al., 1998). We have modified MMPacking to handle nodes with varying bandwidths and storage capacities. We have also modified the original algorithm to reduce the cost of adapting the server to changes in client demand. For example, even a small change in demand can cause MMPacking to move every file from one node to another. In contrast, our algorithm attempts to repack files where they were previously located.

Consider a server which is to store $K$ multimedia files, $M_0 \ldots M_{K-1}$ on $N$ nodes, $S_0 \ldots S_{N-1}$. Each file $M_i$ has demand $D_i$, each node $S_j$ has bandwidth $B_j$ and it is assumed that $K \gg N$.

We begin by evaluating the *target cumulative demand*, $G_j$ for each node $j$, which represents the bandwidth of the node as a proportion of the total server bandwidth:

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

We then define for each node the target shortfall, $H_j$, which is the difference between the cumulative demand for the files packed on the node and the target cumulative demand. As the packing algorithm executes and files are assigned to nodes, this value will decrease. Formally, if the cumulative demand for the files assigned to node $j$ is $C_j$, then the target shortfall is $H_j = G_i - C_j$.

The assignment of files to nodes takes place in rounds. The nodes are sorted by *decreasing* target shortfall, $H_j^l$, at the beginning of each round $l + 1$ of assignments and the files are initially sorted by *increasing* demand. During each round of file assignments, we begin with the first node. The file selected for assignment to that node is the first file in the list *that was previously stored on the same node*. If no suitable file exists, then the first file on the list is assigned to the node. After assigning a file to a node during round $l$, the target shortfall, $H_j^l$, of the node is reduced by the demand associated with the assigned file. If assigning the file to the node completely satisfies its demand, the file is removed from the list of files remaining to be assigned. However, if the demand for the file exceeds the target shortfall for the node, the demand for the file is decreased by the node's target shortfall and the file remains on the list. In this case, the node is removed from the list of nodes, preventing further assignment of files to that node. The assignment of files to nodes is illustrated in Figure 2. A round of assignments ends if the target shortfall of the current node is still greater than the target shortfall of the next node in the list, or if the end of the list has been reached. This is required to force replication for only the most demanding files.

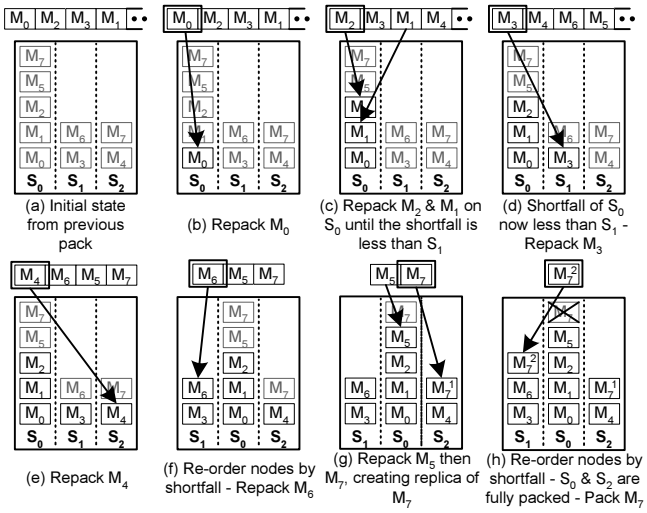


Figure 2: Basic Dynamic RePacking Algorithm

Every period of length $\tau$, the Dynamic RePacking algorithm described above needs to be executed by one of the HammerServers to determine if any change is required to the current assignment of files to nodes. Since each HammerServer will see the same ordered list of members of the Maestro server group, we simply select the first member in the group to evaluate the file assignment. The algorithm is executed using a snapshot of the replicated server state, resulting in a list of files to be either removed from nodes or added to them. A list of required changes is sent to each HammerSource.

**Client Request Redirection**

To perform load-balancing, clients must be directed to the least-loaded node in the cluster. Our prototype server uses the RTSP protocol (Schulzrinne et al., 1998), making the redirection of requests trivial. When an RTSP request is received, the most suitable node to supply the stream is evaluated and the HammerServer sends an RTSP redirect response back to the client, redirecting it to the selected node. The initial RTSP requests are distributed among HammerServer instances using a commodity network load-balancing solution.

**PERFORMANCE**

In this section, we provide early results from our prototype HammerHead server implementation. Detailed results of a simulation of the Dynamic RePacking policy can be found in our previous work (Dukes and Jones, 2002).

To perform the tests, we constructed a four node cluster. A workload generator was developed to generate client requests and receive the resulting streams. Request inter-arrival times were generated from an exponential distribution. Individual files were requested with a frequency determined from a Zipf distribution with the parameter $\theta = 0$ as described by Chou et al (Chou et al., 2000).

Table 1: Server Test Parameters

| | Test 1 | Test 2 | Test 3 | Test 4 |
|---|---|---|---|---|
| Number of nodes ($N$) | 4 | | | |
| Number of files ($K$) | 100 | | | |
| File size (MB) | 3.1 | | | |
| File bit-rate (Kbps) | 266 | | | |
| Stream duration (seconds) | 94 | | | |
| Node storage (MB) | unrestricted | | | |
| Node bandwidth (Kbps) | 25600 | | | |
| Evaluation period (seconds) | 200 | | | |
| Number of sliding window periods | 8 | | | |
| Simulation time (hours) | 2 | | 10 | |
| Popularity rotation period (hours) | – | | 1 | |
| Requests per hour | 6000 | | | |

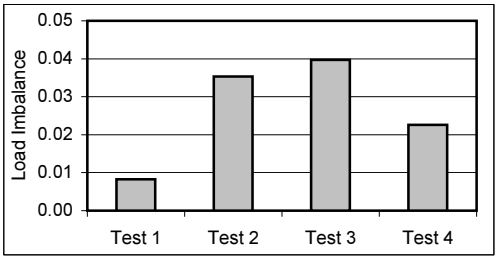


Figure 3: Server Load Imbalance

We performed four tests. Each test used a set of 100 uniform files, with identical duration, bit-rate and size. In the first test, we placed a copy of every file on every node and did not perform Dynamic RePacking. This allowed us to determine a performance baseline for future tests. In the second test, the set of files was assigned to the server nodes in round-robin order and Dynamic RePacking was enabled. In the third test, we periodically rotated the popularity of the files, so the least popular file would suddenly become the most popular, and the most popular files would gradually become less popular. This test was designed to examine the server's response to changing client behaviour. For our final test, we repeated test three, but altered the Dynamic RePacking policy to create a minimum of two copies of the ten most popular files, giving greater flexibility to perform load balancing at the expense of increased storage utilisation. The parameters for each of the four tests are summarised in table 1.

In each of the tests, we recorded the bandwidth utilisation of each node every 15 seconds. For each set of samples, we used the standard deviation of the bandwidth utilisation as an expression of the degree of load-balancing at the time when the samples were taken. The average of this value over the duration of each test is shown in Figure 3. Figure 4 shows the average storage requirement (in units of files) for each test.

Test one achieves the best load-balancing, but at the expense of a high storage capacity overhead, since each node stores the entire set of files. Since most client requests are for a small number of popular files, replicating the remaining unpopular files is wasteful. Test one is our performance baseline.

In test two, with Dynamic RePacking enabled, the load imbalance is only slightly higher than that for the baseline configuration. On average, there was a standard deviation of 3.5% in node utilisation, compared with 0.8% for the baseline. The advantage of selective replication is clear from Figure 4 – the storage capacity required for test two was approximately 25% of that for test one, since only a small number of files are replicated.

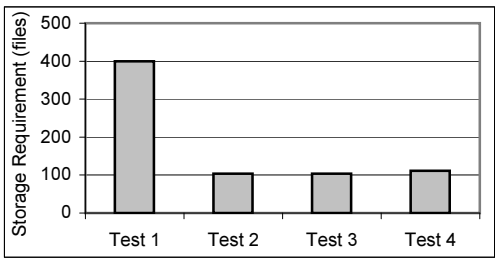


Figure 4: Server Storage Requirement

The load imbalance for test three is slightly higher than that for test two (4.0%), illustrating the cost of adapting the server to changes in file popularity over time. The storage capacity requirement is similar to that for test two.

Finally, in test four, where we created at least two copies of the most popular 10% of the files, the load-balancing is significantly improved, with an average 2.2% standard deviation in node utilisation. This improvement is at the expense of only a marginal increase in the average storage requirement.

In summary, Dynamic RePacking, as implemented in the HammerHead server, achieves a level of load balancing that is only slightly less than the ideal baseline performance, while significantly reducing the storage requirement. The storage cost for the baseline configuration is proportional to $N \times K$, where $N$ is the number of cluster nodes and $K$ is the number of files. The minimum storage cost, which may be achieved using a cluster based on striping without fault-tolerance, is proportional to $K$. Both the tests described here and a simulation study of our Dynamic RePacking policy (Dukes and Jones, 2002) have shown that the storage cost for our policy is approximately proportional to $K + 2N$, which is significantly better than the baseline configuration.

## RELATED WORK

Several dynamic replication policies have been proposed in the past, however, we feel that none of these existing policies were suitable for the HammerHead multimedia server cluster. The DASD dancing policy (Wolf et al., 1995) attempts to assign file to nodes such that the ability of the server to move active streams from one node to another is maximised, without performing accurate static load-balancing based on expected demand. The Bandwidth to Space Ratio (BSR) policy (Dan and Sitaram, 1995b) attempts to balance the ratio of used bandwidth to used storage across server nodes. Our policy assumes bandwidth alone is the primary constraint and storage capacity is only considered as a secondary constraint. Another existing policy (Venkatasubramanian and Ramanathan, 1997) creates the most replicas of the least demanding files. In contrast, our policy creates the most replicas of the most demanding files. In-

tuitively, this increases the ability of the server to perform load balancing as client requests arrive. Another policy, called Dynamic Segment Replication (DSR) (Dan and Sitaram, 1995a) creates partial replicas of files and may be used to complement our Dynamic RePacking policy. Another threshold-based replication policy (Chou et al., 2000) uses threshold values to determine if replication is required for a particular file. Our simulation results from a previous study (Dukes and Jones, 2002), however, suggest that this leads to higher storage utilisation than our policy. Finally, the MMPacking policy (Serpanos et al., 1998) has been used as the basis for our Dynamic RePacking policy.

Group communication systems have been used before to manage multimedia server clusters (Anker et al., 1999). However, the focus of this work was on fault-tolerance, rather than replication and load-balancing. This work would complement our own.

## CONCLUSIONS

Software for providing multimedia streaming services using commodity hardware is widely available. However, the use of such software in a cluster environment presents us with specific problems. If we assume that replicating all multimedia content on every server node will be prohibitively expensive, then we need to distribute the content among the nodes, without compromising the scalability and availability of the server.

In this paper, we have described the HammerHead multimedia server cluster and, in particular, its use of the Dynamic RePacking policy to perform selective replication of content and provide load-balancing. The HammerHead architecture has been designed to integrate with existing multimedia server software, providing a cluster-aware layer which is responsible for estimating the demand for each file, distributing content among server nodes and redirecting client requests to suitable server nodes. We have used the Ensemble group communication toolkit to provide reliable communication between server components and cluster membership detection.

Test results indicate that the HammerHead server compares favourably with our chosen baseline – a server cluster that replicates all content on every node. We significantly reduced the storage cost for the server, with only a small increase in load-imbalance. The performance results presented in this paper represent preliminary results and further, extensive performance analysis is required.

We are also undertaking further development of the server. In particular, we are investigating server clusters with more complex storage hierarchies, how these hierarchies might be reflected in the server state information and the resulting impact on the use of Dynamic RePacking.

## ACKNOWLEDGEMENTS

The research described in this paper is part funded by Microsoft Research, Cambridge, UK.

## REFERENCES

Anker T., Dolev D. and Keidar I., 1999, "Fault Tolerant Video on Demand Services". In *Proceedings of the 19th International Conference on Distributed Computing Systems*, Austin, Texas, USA.

Bolosky W.J., Fitzgerald R.P. and Douceur J.R., 1997, "Distributed Schedule Management in the Tiger Video Fileserver". In *Proceedings of the Sixteenth ACM Symposium on Operating System Principles*, Saint-Malo, France, 212–223.

Chou C., Golubchik L. and Lui J., 2000, "Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication". In *Proceedings of 20th International Conference on Distributed Computing Systems*, Taipei, Taiwan, 64–71.

Dan A. and Sitaram D., 1995a, "Dynamic Policy of Segment Replication for Load-Balancing in Video-on-Demand Servers". *ACM Multimedia Systems*, 3, no. 3, 93–103.

Dan A. and Sitaram D., 1995b, "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*, San Jose, California, USA, 376–385.

Dukes J. and Jones J., 2002, "Dynamic RePacking: A Content Replication Policy for Clustered Multimedia Servers". In *Proceedings of the Microsoft Research Summer Workshop*, Cambridge, England.

Guha A., 1999, "The Evolution to Network Storage Architectures for Multimedia Applications". In *Proceedings of the IEEE Conference on Multimedia Computing Systems*, Florence, Italy, 68–73.

Hayden M. and van Renesse R., 1997, "Optimising Layered Communication Protocols". In *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*.

Lee J.Y.B., 1998, "Parallel Video Servers: A Tutorial". *IEEE Multimedia*, 5, no. 2, 20–28.

Little J.D.C., 1961, "A Proof of the Queuing Formula: $L = \lambda W$". *Operations Research*, 9, no. 3, 383–387.

Patterson D.A., Gibson G. and Katz R.H., 1988, "A Case for Redundant Arrays of Inexpensive Disks (RAID)". In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, USA, 109–116.

Schulzrinne H., Rao A. and Lanphier R., 1998, "Real Time Streaming Protocol (RTSP)". IETF RFC 2326 (proposed standard), available at http://www.ietf.org/rfc/rfc2326.txt.

Serpanos D.N., Georgiadis L. and Bouloutas T., 1998, "MMPacking: A Load and Storage Balancing Algorithm for Distributed Multimedia Servers". *IEEE Transactions on Circuits and Systems for Video Technology*, 8, no. 1, 13–17.

Vaysburd A., 1998, *Building reliable Interoperable Distributed Objects with the maestro Tools*. Ph.D. thesis, Cornell University.

Venkatasubramanian N. and Ramanathan S., 1997, "Load Management in Distributed Video Servers". In *Proceedings of the International Conference on Distributed Computing Systems*, Baltimore, Maryland, USA.

Wolf J.L., Yu P.S. and Shachnai H., 1995, "DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems". In *Proceedings of ACM SIGMETRICS '95*, Ottawa, Ontario, Canada, 157–166.

Wong P.C. and Lee Y.B., 1997, "Redundant Arrays of Inexpensive Servers (RAIS) for On-Demand Multimedia Services". In *Proceedings ICC 97*, Monréal, Québec, Canada, 787–792.