A Demand Adaptive and Locality Aware (DALA) Streaming Media Server Cluster Architecture *

Zihui Ge   Ping Ji   Prashant Shenoy

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

{gezihui,jiping,shenoy}@cs.umass.edu

ABSTRACT

The wide availability of broadband networking technologies such as cable modems and DSL coupled with the growing popularity of the Internet has led to a dramatic increase in the availability and the use of online streaming media. With the "last mile" network bandwidth no longer a constraint, the bottleneck for video streaming has been pushed closer to the server. Streaming high quality audio and video to a myriad of clients imposes significant resource demands on the server. In this work, we propose a demand adaptive and locality aware (DALA) clustered media server architecture that can dynamically allocate resources to adapt to changing demand and also maximize the number of clients serviced by the server cluster. Moreover, our design exploits temporal locality among requests by dispatching newly arriving requests to servers that are already servicing prior requests for those objects, thereby extracting the benefits of locality. We explore the efficacy of the DALA clustered architecture using simulations. Our simulation results show that DALA is highly adaptive, exhibits significant performance gains when compared to static schemes, and has a low system overhead. Our results demonstrate that DALA is a simple, yet effective approach for designing clustered media servers.

Categories and Subject Descriptors

Computer Systems Organization [**PERFORMANCE OF SYSTEMS**]: Design studies

General Terms

Design

Keywords

Streaming Media, Server Cluster, Locality Aware, Demand Adaptive

---

1. INTRODUCTION

The wide availability of broadband networking technologies such as cable modems and DSL coupled with the growing popularity of the Internet has led to a dramatic increase in the availability and the use of online streaming media such as entertainment movies, news clips, and educational and training materials. With the "last mile" network bandwidth no longer a constraint, the bottleneck for video streaming has been pushed closer to the server [24]. Streaming high quality audio and video to a myriad of clients imposes significant resource demands on the server. One cost-effective approach to scale the server capacity is to employ a *server cluster*. For instance, a clustered approach is commonly used to design scalable web servers and several research prototypes and commercial products have been developed with such an architecture [1, 18]. However, there are important differences between web workloads and streaming workloads that prevent these approaches from being directly used for clustered streaming media servers. For instance, clustered web servers typically replicate web content on each server, thereby allowing any server to service an incoming request. In contrast, streaming media objects are several orders of magnitude larger than web objects and repositories of high-end streaming servers can exceed several terabytes. Hence, full replication of content at each server is prohibitively expensive, necessitating a partitioning of the content and the workload among servers in the cluster. Moreover, retrieval and transmission of streaming media objects imposes real-time constraints, while web requests are typically serviced in a best-effort manner. Further, the service time of web requests is in the order of hundreds of milliseconds, while a typical streaming session lasts tens of minutes. Due to the long-lived nature of sessions and real-time constraints, clustered streaming servers need to be designed differently from clustered web servers. On the other hand, like web workloads, requests for streaming media are usually skewed and unpredictable. Accesses to streaming media objects follow a Zipf-like distribution [8, 2] and popularities of individual objects tend to vary over time [2]. Hence, dynamic allocation of cluster resources to meet the needs of changing workloads and the dynamic distribution of requests among servers in the cluster are important design issues in clustered streaming servers.

To address these challenges, in this paper, we propose a demand adaptive and locality aware (DALA) clustered media server architecture that can dynamically allocate resources to adapt to changing demand and also maximize the number of clients serviced by the server. The DALA architecture is *demand adaptive* since it can dynamically vary the amount of resources (number of servers, disk space, disk and network bandwidth on each server) allocated to each object based on its popularity—more resources are
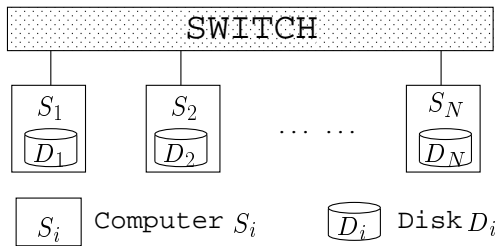
Figure 1: A Sample Cluster Server Infrastructure

Figure 2: Conceptual DALA Server Cluster Design

allocated to objects with increasing popularity and these resources are relinquished as the popularity wanes. Moreover, the architecture is *locality aware* since it can dispatch newly arriving requests to servers that are already servicing prior requests for those objects and thereby extract the benefits of memory caching. Our resource allocation architecture and request distribution mechanisms attempt to optimize the utilization of the memory, disk and network resources within the cluster, thereby maximizing overall system capacity. We demonstrate the efficacy of the DALA clustered architecture using simulations. Our simulation results show high responsiveness to changing loads, performance gains due to locality awareness, and low overheads.

The remainder of this paper is organized as follows. Section 2 presents the details of our DALA clustered server architecture. Section 3 presents our experimental results. Section 4 discusses related work, and finally, Section 5 presents some concluding remarks.

2. DALA SERVER CLUSTER ARCHITECTURE

In this section, we present the details of our demand-adaptive and locality-aware architecture for clustered streaming servers. We first present the system model assumed in our work and then present the details of our architecture.

2.1 System Model

Consider a cluster of N servers, denoted by $S_1, S_2, ..., S_N$, that are interconnected by a high-speed switch (see Figure 1). Assume that each server S_i has a local disk D_i for storing streaming media content.

Our architecture assumes that each server can serve multiple objects from its local disk and that each object can be served from multiple servers. The number of servers that can serve a streaming media object depends on its popularity — the more popular the object, the larger is the number of servers allowed to serve it. We refer to a set of servers that can serve a streaming media object as a *group* and use G_j to denote the group that serves object j. Consider the example shown in Figure 2 where each circle denotes a group and each square denotes a server. The figure illustrates the difference in group sizes, the overlap between groups, and the variation in group sizes with changing popularities (see Fig 2(a) and (b)). We require that each group have at least one server in it and that the server be the one holding the primary copy of the object.

The local disk on each server consists of two partitions — one used to store primary copies of objects and the other used to store dynamically created replicas of objects.

- *Primary copy partition:* At least one copy of each object is stored permanently on some server in the cluster. This statically stored file is referred to as the primary copy of the
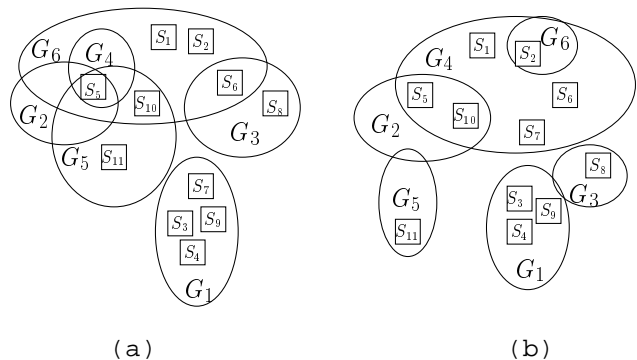
object. Our current design assumes a single primary copy for each object and we define

r_j: as the server that has the primary copy of object j

where $1 \leq j \leq M$, and M is the total number of objects.

- *Dynamic service partition:* When a group consists of multiple servers, each server (with the exception of r_j) stores a replica of the object on its local disk. These replicas are stored in the dynamic service partition. The storage space for a replica is reclaimed when the server leaves the corresponding group.

The relative sizes of the primary copy and the dynamic service partitions depend on the mean group size across all objects in the system. Assuming a mean group size of k, a good rule of thumb is to assign storage space in the ratio $1 : k - 1$. We assume that primary copies of objects are assigned to servers in a round-robin manner. Initially, the dynamic service partition is empty (all objects have an initial group size of 1). Our architecture also assigns a token for each object. The token holder for an object is responsible for accepting and processing all new requests for that object. We define

t_j: as the token holder for object j

Initially we set $t_j = r_j$ (the server with the primary copy is assigned to be the initial token holder for the object).

2.2 DALA Demand Dissemination Protocol

Consider a cluster of N servers as described in Section 2.1. Assume that a new request for a streaming media object arrives at one of these servers. We assume that each server maintains information about the token holder t_j for all objects stored in the cluster. When a new request arrives, the server determines the token holder for the object and forwards the request to the token holder for further processing (an alternate approach is to maintain this knowledge at a layer-7 switch, which then forwards the incoming request directly to the corresponding token holder).

Upon receiving a new request, the token holder performs admission control to determine whether sufficient resources exist to service the new request locally. Such an admission control algorithm needs to ensure that sufficient network bandwidth, disk bandwidth and memory buffers exist to service the new request. A number of admission control algorithms have been proposed recently [12, 7, 23]; any such algorithm suffices for this purpose. Depending on the results of admission control, the token holder either accepts the request or decides to pass the token to another server in the cluster
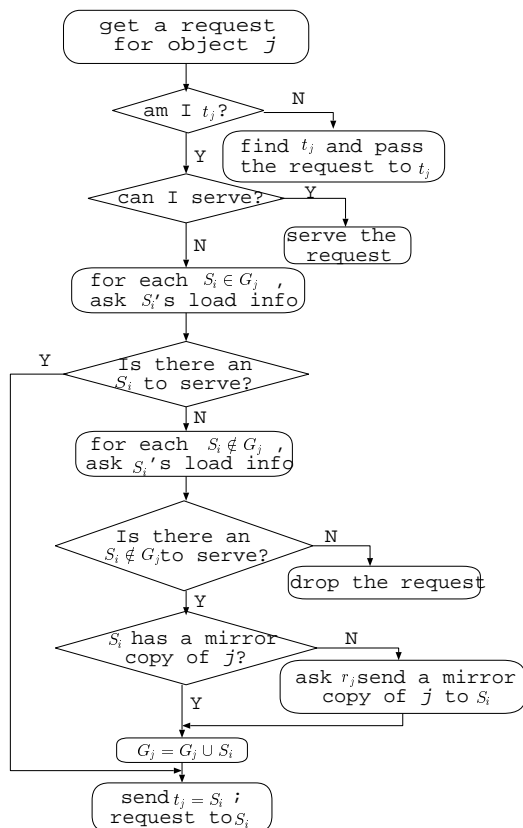
2

get a request
for object j

am I t_j?

N

find t_j and pass
the request to t_j

Y

can I serve?

Y

serve the
request

N

for each $S_i \in G_j$,
ask S_i's load info

Is there an
S_i to serve?

Y

N

for each $S_i \notin G_j$,
ask S_i's load info

Is there an
$S_i \notin G_j$ to serve?

N

drop the request

Y

S_i has a mirror
copy of j?

N

ask r_j send a mirror
copy of j to S_i

Y

$G_j = G_j \cup S_i$

send $t_j = S_i$;
request to S_i

Figure 3: A Pictorial Representation of the DALA Demand Dissemination Algorithm

(which is then responsible for determining how to proceed with the request).

- If admission control is successful, then the token holder t_j accepts the request and services it.

- If admission control fails, then the token holder lacks sufficient resources to service any further requests for this object. Hence, the token holder invokes the token passing protocol (described in Section 2.3) to determine a new token holder for the object. The request is then forwarded to the new token holder for processing (and subsequent requests for the object are also sent to the new token holder).

- If the token passing algorithm is unable to locate a new token holder in the group, then all group members are overloaded indicating that the group needs to be expanded. In such a scenario, the token holder invokes the dynamic group membership protocol (described in Section 2.4) to pull a new member into the group. Both the token and the request are forwarded to this server for further processing.

- If no suitable servers can be pulled into the group, then cluster lacks resources to service the request and the request is denied.

Figure 3 illustrates this demand dissemination protocol pictorially. Next we briefly describe the token passing and dynamic group membership protocols.

2.3 Token Passing Within a Group

The token holder for an object processes and services newly arriving requests until no additional requests can be accepted. At this point, the token is passed to another server in the group. To do so, the token holder queries group members for their load information and picks the server with the least load. Assuming that this server is willing to assume token holder responsibilities, the token is sent to this server and its identity is broadcast to the entire cluster as the new token holder for the object. If all group members are heavily loaded (indicating that no group member can assume token holder responsibilities), then the dynamic group membership protocol is invoked to expand the group (see Section 2.4).

Several design decisions affect the efficiency of the token passing protocol. First, choosing a new token holder requires knowledge of the load on each server in the group. The load information can be gathered in an eager or lazy fashion. In the former approach, each server periodically broadcasts its load information to all other servers. Each server maintains a table of the load on various servers and uses this table to pick a new token holder when the need arises. Since the recency of the load information determines its accuracy, such an approach can result in decisions based on inaccurate information [10]. Further, if token passing is not too frequent, exchanging load information results in wasted messages. The advantage though is that a new token holder can be chosen quickly, reducing the latency of token passing. The lazy approach, on the other hand, queries each server for its load information only when necessary (at token passing time). While this results in up-to-date load information, it can increase the latency of token passing (since the token holder must wait until all servers in the group have responded before making a decision or time out). A simple but effective approach to improve the latency of lazy load gathering it to use randomization. Theoretical studies have shown that picking two (or a constant k, $k > 1$) servers at random and choosing one of the two (or k) servers based on their loads is as effective as querying all servers and picking the one with minimum load [17]. This is especially true when the group size is large. Such a randomized approach reduces the complexity of lazy token passing to a constant and makes it independent of group size.

A second design decision is the invocation of the token passing algorithm itself. This can also be done in a lazy or eager manner. In lazy token passing, a new token holder is chosen only when admission control for a new request fails. In eager token passing, a new token holder is chosen when the load on the server exceeds a high threshold. Whereas lazy token passing can increase the latency for servicing new request, eager token passing can increase the overhead of unnecessary token passes.

As a final caveat, we note that handling lost tokens in DALA is similar to handling the failure of a coordinator within a group of distributed processes [22]. Group members can check on the status of the token holder by exchanging heartbeat messages. In the event that the token holder is down, a new token holder can be chosen by running a simple leader election algorithm within the group [22]. Such a strategy fails only when the token holder was the sole member of the group (and has failed) and no other copy of the video exists in the cluster; in such cases, the video can no longer be served by the cluster until a copy is retrieved from tertiary storage.

2.4 Adapting the Group Size to the Load

To accommodate changing workloads, the group size of an object is varied dynamically based on its popularity. The group size is increased when the object popularity increases and is shrunk when the popularity wanes. This is achieved by two separate procedures that constitute our dynamic group membership protocol —

the *group expansion* procedure and the *group contraction* procedure.

A group is expanded when the token passing protocol fails to pick a new token holder from the existing group. To add a new member to the group, the token holder needs to know the load on each remaining server and whether these servers already have a replica of this object (since storage space in the dynamic service partition is reclaimed in a lazy fashion and only to make room for a new incoming replica, a server might continue to store a copy of this object from its previous incarnation as a group member for the object). Given this information, the current token holder picks the server with the least load and preferably one that has a replica of the object. This server is invited to join the group and also sent a copy of the object if it doesn't have one already. The server is then sent the token for the object and the pending request for further processing. The identity of the new group member is broadcast to the remaining group members and its identity as the new token holder is sent to the entire cluster. Like the token passing protocol, the group expansion procedure can be invoked in a lazy or eager manner. This design decision has important implications on performance and the startup latency for requests, especially since propagation of a replica to a new group member can involve significant latency (due to large object sizes). On the other hand, eager group expansion can result in wasted copying effort.

The group size for an object shrinks when its popularity decreases. Our architecture uses a timeout strategy for group contraction — after finishing the service of all the related requests, if a group member does not receive the token within a timeout interval, it automatically relinquishes group membership (and announces this to the rest of the group). This allows servers to free up resources and reuse them for objects with increasing popularities.

2.5 Exploiting Locality Awareness

One possible approach for load balancing within a cluster is to distribute incoming requests evenly among group members. Such a request distribution policy, however, does not allow the cluster to exploit temporal locality among requests. The need to exploit locality among requests motivates our decision to use a token for each object. By employing the concept of a token holder, all incoming requests for an object are sent to the same server (the token holder), enabling the server to exploit temporal locality relationships among requests. Each server employs an interval caching policy [11] and sending all request to the token holder improves the chances of an interval formation, resulting in substantial performance gains for very popular objects (each interval that is formed allows the server to service one or more streams directly from memory buffers, thereby saving precious disk bandwidth).

Similar locality-aware request distribution policies for web requests have been studied in [18] and have exhibited substantial performance gains. Since web content is served in a best-effort manner, in such servers, exploiting locality needs to be weighed against load balancing considerations (load balancing and locality awareness have opposite tradeoffs). Streaming media servers, on the other hand, provide guaranteed service where the quality of service is ensured through admission control. Thus, we are able to exploit locality to the maximum extent without worrying about load balancing considerations (so long as QoS requirements are not violated, the exact distribution of load among various servers is of secondary importance). Hence, given the long-lived nature of streaming sessions, we expect to observe significant performance gains using a locality-aware request distribution policy.

3. EXPERIMENTAL RESULTS

We evaluate the efficiency of the DALA architecture by using simulations. In this section, we present our experiments' settings and the corresponding evaluation results.

3.1 Effect of Demand Adaptation

Our first experiment simulates a flash flow scenario where the popularity (and request frequency) of an object increases suddenly, stays at that level for a certain period, and then drops gradually. The objective of the experiment is to examine how the DALA architecture adjusts to unpredictable workload changes.

Our simulation setup assumes a cluster of 10 low-end commodity PCs that constitute the DALA cluster. Each server is assumed to have 128 MB of memory, a 20 GB hard disk with a maximum throughput of 40 Mbps, and a 100 Mbps Ethernet card. We assume that the cluster serves 100 streaming media objects and that the length of these objects is uniformly distributed between 60 and 120 minutes. Each object is assumed to be MPEG-1 encoded with a maximum bit rate of 1.5 Mbps.

Figure 4 depicts the workload seen by the cluster over a 12 hour period. Request arrivals are assumed to be Poisson. The top solid curve shows the total number of request arrivals over 5 minute intervals, while the dashed curve below it plots the number of requests for the most popular object in the cluster. The two flat lines represent the total network bandwidth capacity and the total disk bandwidth available in the cluster.

We show how DALA adapts to workload changes by examining how the group size G_1 varies over time (G_1 is represented by the bottom curve in the figure). As shown, as the workload increases, G_1 increases in steps until the group spans all 10 servers in the cluster. When the workload starts decreases after the first six hours, we see that G_1 contracts gradually as well. Observe that there is a time lag between the decrease in workload and the corresponding decrease in G_1. This is because, the a server drops out of a group only after it has finished servicing all ongoing requests for the object plus a timeout duration.

The set of crosses in the figure represents the number of requests that are denied. Observe that the most of denied requests appears after the group size G_1 reaches its maximum possible value of 10 servers. This indicates that DALA makes a very good use of system resources. Further, observe that requests are dropped only when the workload approaches (and, in some cases, exceeds) the total disk and network capacities of the cluster (request drops are inevitable if the total workload exceeds total capacity).

Together these results show that DALA adapts to changing workload and makes judicious use of cluster resources.

3.2 Comparison With other Request Distribution Policies

To further evaluate the performance benefits due to demand adaptation, we compare DALA to two other request distribution policies: *static resource allocation (SRA)* and *static resource allocation with locality (SRL)*. In static resource allocation, the number of servers allocated to serve each object is determined based on long-term popularities for each object. Thus the configuration of the system is done periodically and manually; between configuration changes, the number of servers assigned to each object (i.e., the group size) is kept fixed. When a request arrives, it is sent to a randomly chosen server in its group. Static resource allocation with locality (SRL) is similar to SRA, except that it exploits locality among request arrivals. This is achieved using the concept of a token holder and token passing. Thus, SRL is like DALA, except that the group sizes are fixed.
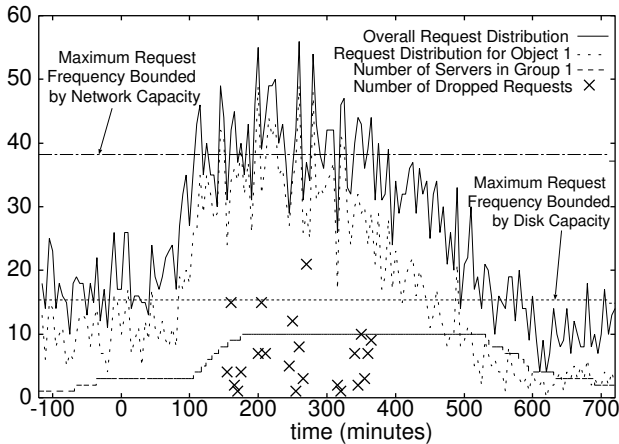
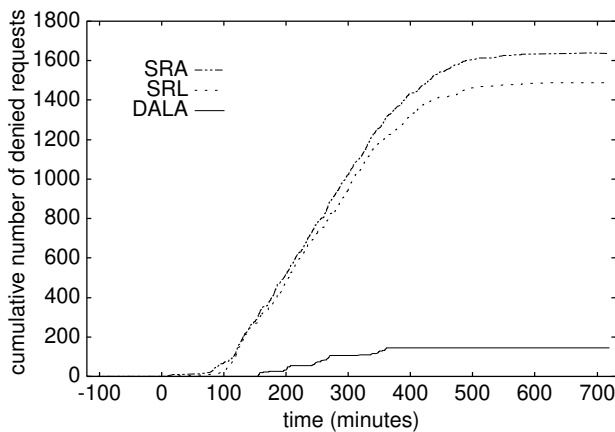**Figure 4: Effect of Dynamically Changing Workloads.**

Figure 6: Comparing the Capacity of Server Clusters

Figure 5: Comparing Different Request Distribution Policies

We use the same setup as our previous experiment to evaluate the three policies, SRA, SRL and DALA. To determine the group sizes for SRA and SRL, we first run the system with DALA for about 2 hours (120 minutes) and then use the group sizes chosen by DALA to configure SRA and SRL. The group sizes remain fixed for the two schemes for the rest of the simulation. Choosing the group size in this manner permits a fair comparison among the three policies. Using the same workload as our previous experiment, we compute the number of requests denied by the three policies. Figure 5 plots the cumulative number of dropped requests for the three policies. The difference between SRA and SRL indicates the benefits of locality awareness, while that between SRL and DALA indicates the additional benefit due to demand adaptation. The figure shows DALA significantly outperforms SRA and SRL due to its demand adaptive and locality aware nature.

3.3   Effect of Changing Workloads

Whereas the workload in the previous section was dominated by requests to a single popular object (to better illustrate some of the key aspects of the DALA architecture), in this section, we evaluate the performance of DALA over a wide range of workloads.

To do so, we assume a cluster of 20 high-end servers, each with 1GB memory, a high-end 100 GB disk with a maximum throughput
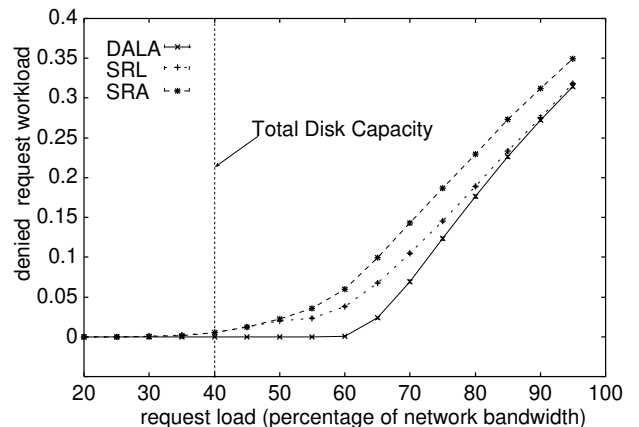
of 400 Mbps and a 1 Gbps Ethernet card. The cluster is assumed to serve 300 high-quality MPEG-2 videos, each ranging from 60 to 120 minutes and with encoding rates ranging from 4 to 8 Mbps.

Like in the previous experiment, request arrivals are assumed to be Poisson and request popularities follows Zipf's distribution. To simulate changing popularities, we randomly pick a small fraction (10%) of the objects after every 100 requests and exchange their ranks. We vary the arrival rate and measure the corresponding ratio of the dropped request load to the total request load. For each level of request arrival rate, we generate request traces of 24 hours long and repeat each experiment for DALA, SRA and SRL. Like in the previous experiment, the group sizes for SRA and SRL are determined using DALA during the warmup phase, which is set to be the first half (12 hours) of each experiment. The measurement starts when the warmup phase is over.

Figure 6 plots the observed drop rate for the three policies for different loads (the X axis plots the load as a percentage of the total network capacity; the total disk capacity is 40% of the network capacity). For each value of request arrival rate, we run 100 experiments and calculate the mean and 95% confidence interval of the dropped request load ratio. We are actually showing the confidence intervals in Figure 6. However, they are too small to be easily distinguished.

Since SRA does not exploit any locality, it gets minimal benefits from memory caching. Hence, as shown in Figure 6, the capacity of the cluster with SRA is limited by the total disk capacity. Beyond this point, SRA starts dropping requests and the drop rate increases with increasing arrival rates. SRL has a lower drop rate than SRA due to benefits from memory caching. DALA yields the best performance. DALA starts dropping requests only when the load exceeds 60% of the total network capacity; this load is around 150% of the total disk capacity. Thus, the demand adaptive and locality aware nature allows DALA to support 50% additional clients beyond what the disk can support. At very high arrival rates, all server resources are almost fully utilized regardless of the policy. This explains the negligible difference between DALA and SRL; the difference between SRL and SRA in this region shows how much the system capacity can be increased by exploiting request locality.

Finally, Figure 7 depicts the overheads of DALA for different workloads. The figure shows the number of control operations, namely token passes and group expansions, per minute at different loads. As of studying the dropped workload, we also calculate the
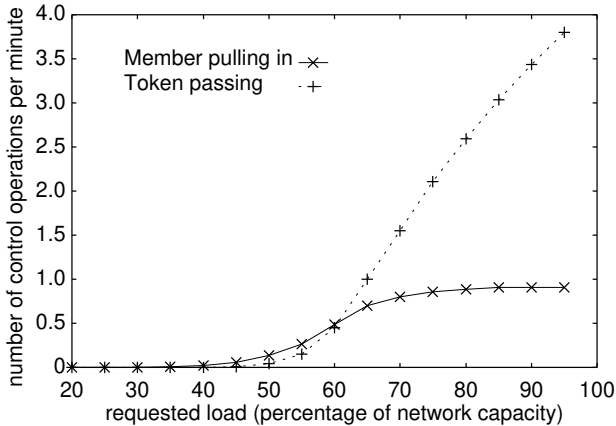
Figure 7: Control Overhead of DALA

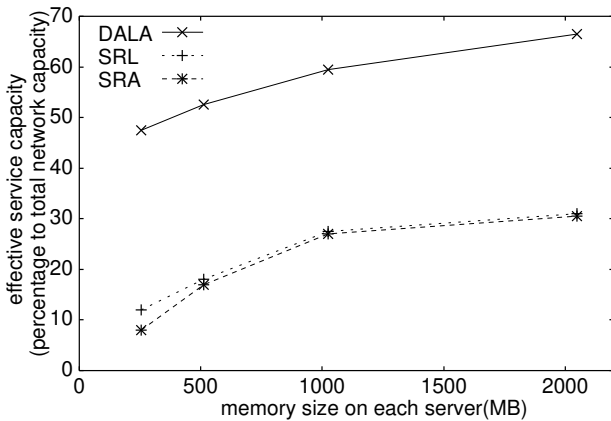**Figure 9: Effective Service Capacity of the Server Cluster**

Figure 8: Effective Service Capacity of the Server Cluster

mean and 95% confidence interval per 100 sample runs in studying the system overhead, and we show the confidence intervals in Figure 7 as well. We observe from this figure that, even at very heavy loads, there are fewer than five control operations per minute, suggesting a very low control overhead.

3.4 Determining Effective Service Capacity

In many scenarios, we are interested in the maximum workload the system can support before it starts dropping requests. This workload indicates the effective service capacity of the server cluster. In this section, we use this metric to evaluate the system performance under different system parameters.

To obtain the effective service capacity of the cluster, we conducted an experiment where we gradually increased the request arrival rate (and hence the workload) from 0 to 100% in steps of 0.5% (here, a load of 100% represents the normalized load at with the network interface on a node saturates). At each workload level, we conducted 30 experiments and used the hypothesis test to examine whether the ratio of denied workload is zero. If the ratio of denied requests at a given workload level is significantly different from zero (at a significance level of 0.05), then we stop and take the last successful workload level as the effective service capacity of the server cluster.

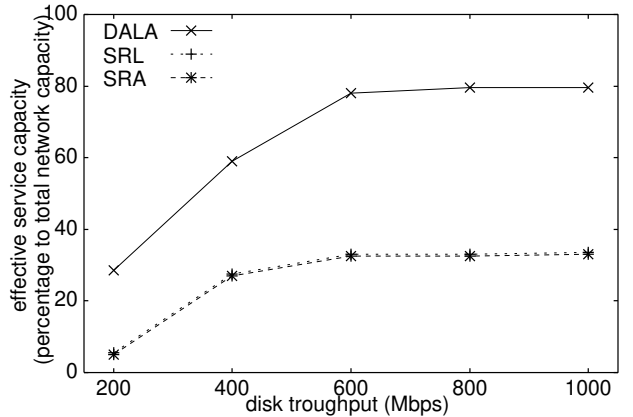In the first set of simulations, we use the same video repository

as described in Section 3.3 and also use the same server cluster configuration except that we vary the size of memory on each server. Figure 8 compares the effective service capacities of different architectures when the memory sizes on each server are 256 MB, 512 MB, 1 GB and 2 GB, respectively. In all cases, DALA shows a significantly higher effective service capacity than SRL or SRA. Also, SRL and SRA do not exhibit a significant difference in their effective service capacities, especially when memory size on each node is sufficiently large. This can also be inferred from Figure 6, since exploiting temporal locality only improves the system performance at high loads or during overloads. For all mechanisms, we see that the effective service capacity increases sub-linearly as more memory added to each server.

Figure 9 shows another set of experiments where each server in the cluster has a fixed 1GB memory and we vary the disk throughput on each server. Consistent with previous experiment, DALA shows a substantial performance gain over SRL or SRA. We also observe that when the disk throughput is low (less than 600Mbps), the disk is the bottleneck device. Hence, the system performance improves with increase in the disk throughput. Beyond a throughput of 600Mbps (with a network interface bandwidth of 1 Gbps), we see no further gains by further increasing the disk throughput. This is because the effects of memory caching govern the system performance in this operating region and increasing the disk throughput can no longer help the system achieve a higher effective service capacity.

4. RELATED WORK

Research on video-on-demand systems in the early nineties led to a plethora of efforts on admission control [12, 7], scheduling [23], striping [21, 15, 13] and caching [11]. All of these efforts were primarily targeted toward single node servers.

More recent work has focused on distributed media servers [5, 20, 4, 16]. The Tiger distributed video server employs network striping across a cluster of nodes and uses a centralized controller to coordinate these nodes. Network striping allows the load on various nodes to be balanced. In contrast, DALA stores each object on a single node and uses dynamic replication, token passing and changes to group membership to accommodate changing workloads. Further, unlike the centralized controller in Tiger, DALA uses a fully distributed architecture and has no single point of failure. The Exedra distributed media server also employs distributed striping across disks connected to a storage area network. A set of

transfer nodes access data from disks and transmit them to clients. Some of these techniques are orthogonal to our work and can be employed in DALA as well. For instance, use of a storage area network instead of local disks in DALA can reduce the overheads of tasks such as object replication.

A recent study has focused on algorithms for distributed caching of streaming media content in the Internet [6]. In this approach, objects are partitioned into smaller segments that are replicated across different caches for load balancing. The focus of the work is primarily on placement and caching and is orthogonal to our effort. Moreover, some of the techniques in [6] require long-term popularity statistics, which is not a requirement in DALA.

Finally, several recent efforts have investigated techniques such as periodic broadcast and patching to scale the network capacity of the server to a large number of users using multicast [3, 19, 14, 9]. These techniques complement our work, since all of them can be employed by DALA as well.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a demand adaptive and locality aware (DALA) architecture for clustered media servers. DALA employs a fully distributed architecture and requires no priori knowledge of the workload or object popularities. The resource allocation scheme and request distribution mechanism in DALA optimize the utilization of the memory, disk and network resources within the cluster, thereby maximizing overall system capacity. Our simulation results showed that DALA is highly adaptive, exhibits significant performance gains when compared to static schemes, and has a low system overhead. Our results demonstrated that DALA is a simple, yet effective approach for designing clustered media servers.

As part of future work, we plan to develop analytical models for our techniques and evaluate the efficacy of our architecture using real-world traces. Furthermore, we would like to extend DALA into a distributed server architecture over Internet, which will require us to study the impact of various network parameters on system performance.

Acknowledgments

6. REFERENCES

[1] Nortel alteon web switching. http://www.nortelnetworks.com/products/01/alteon/, 2001.

[2] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of educational media server workloads. In *NOSSDAV'01.*, June 2001.

[3] Kevin C. Almeroth and Mostafa H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal of Selected Areas in Communications*, 14(6):1110–1122, 1996.

[4] Stergios V. Anastasiadis, Kenneth C. Sevcik, and Michael Stumm. Modular and efficient resource management in the exedra media server. In *3rd USNIX Symposium on Internet Technologies and Systems*, San Francisco, California, March 2001.

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

[6] Youngsu Chae, Katherine Guo, Milind M. Buddhilot, Subbash Suri, and Ellen W. Zegura. Silo, rainbow, and caching token: Schemes for scalable, fault tolerant stream caching. *To appear in Special Issue of IEEE JSAC on Internet Proxies.*

[7] E. Chang and A. Zakhor. Cost analyses for vbr video servers. In *Proceedings of Multimedia Computing and Networking (MMCN) Conference*, pages 381–397, 1996.

[8] Maureen Chesire, Alec Wolman, Geoffrey M. Voelker, and Henry M. Levy. Measurement and analysis of a streaming-media workload. In *USITS*, 2001.

[9] Tzi cker Chiueh and Chung ho Lu. A periodic broadcasting approach to video-on-demand service. In *SPIE First International Symposium on Technologies and Systems for Voice, Video, and Data Communications*, volume 2615, Philadelphia PA, Oct. 1995.

[10] Michael Dahlin. Interpreting stale load information. In *19th IEEE International Conference on Distributed Computing Systems (ICDS)*, Austin, TX, May 1999.

[11] A. Dan and D. Sitaram. Buffer management policy for a on-demand video server. Technical Report RC 19347, IBM.

[12] V. Firoiu, J. Kurose, and D. Towsley. Efficient admission control of piecewise linear traffic envelopes at edf schedulers. *IEEE/ACM Transactions on Networking.*

[13] John H. Hartman and John K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, 1995.

[14] Kien A. Hua and Simon Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *SIGCOMM*, pages 89–100, 1997.

[15] J.R.Santos and R. Muntz. Comparing random data allocation and data striping in multimedia servers. In *ACM Sigmetrics*, Santa Clara, CA, June 2000.

[16] J.R.Santos and R.Muntz. Performance analysis of the rio multimedia storage system with heterogeneous disk configurations. In *6th ACM International Multimedia Conference*, Bristol, United Kingdom, Sep. 1998.

[17] Michael Mitzenmacher, Andréa W. Richa, and Ramesh Sitaraman. The power of two random choices: A survey of the techniques and results. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Handbook of Randomized Computing*. Kluwer Press.

[18] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality-aware request distribution in cluster-based network servers. In *ACM Eighth International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, Oct 1998.

[19] Jehan-François Pâris, Darrell D. E. Long, and Patrick E. Mantey. A zero-delay broadcasting protocol for video on demand. In *the Seventh ACM International Multimedia Conference*, Orlando, October 1999.

[20] Olav Sandsta, Stein Langorgen, and Roger Midtstraum. Video server on an ATM connected cluster of workstations. In *International Conference of the Chilean Computer Science Society*, pages 207–217, 1997.

[21] Prashant Shenoy and Harrick M. Vin. Efficient striping techniques for variable bit rate continuous media file servers. *Performance Evaluation*, 38(3):175–199, December 1999.

[22] Andrew S. Tanenbaum. *Distributed Operating Systems*. Prentice Hall, 1995.

[23] Harrick M. Vin, Alok Goyal, and Pawan Goyal. Algorithms for designing multimedia servers. *Computer Communications*, 18(3):192–203, 1995.

[24] Yubing Wang, Mark Claypool, and Zheng Zuo. An empirical study of realvideo performance across the internet. In *ACM SIGCOMM Internet Measurement Workshop*, San Francisco, California, November 2001.