Design and Analysis of A Demand Adaptive and Locality Aware Streaming Media Server Cluster

Zihui Ge[1], Ping Ji[2] and Prashant Shenoy[3]

[1]Adverplex Inc., Wakefield MA, {gezihui@adverplex.com}

[2]John Jay College of Criminal Justice, City University of New York, NY, {pji@jjay.cuny.edu}

[3]University of Massachusetts, Amherst MA, {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 through both performance models and simulations. Evaluation of the models and simulation results show that, DALA exhibits significant performance gains when compared to static schemes. Furthermore, our simulation results show that DALA is highly adaptive, and has a low system overhead. Our results demonstrate that DALA is a simple, yet effective approach for designing clustered media servers.

1 Introduction

The wide availability of broadband networking technologies such as cable modems and DSL, coupled with the popularity of the Internet, has led to a major increase in the availability and the use of online streaming media such as entertainment movies, news clips, and educational and training materials. A concurrent trend has been the dramatic success of video sharing sites such as YouTube and Google Video; these sites allow individuals to upload personal video content, which can then be streamed to any user on the Internet. At the same time, major news sites now offer many news stories in video form, while digital cable companies offer increasingly large repositories of on-demand content. Delivering a significant amount of content to users requires a significant server infrastructure. It is estimated that YouTube serves 40 million videos to users each day, which amounts to 200 TB of served data per day [50]. New content appears daily on VOD servers, on news sites, and on sites such as YouTube. YouTube is an extreme example, where users upload thousands of videos to the site every day. It is well known that popularities of video files are skewed and unpredictable. Accesses to streaming media objects follow a Zipf-like distribution [15, 3], while popularities of individual objects tend to vary over time [3, 13]. In many scenarios, it is not feasible to predict the popularities a priori. For instance, a news web-site can not predict when a major news event will occur, while YouTube can not control what content gets uploaded and which subset of it will become popular among users.

As a result, the underlying server infrastructure needs to be extremely flexible and dynamically allocate resources to meet the needs of dynamic time-varying workloads. One cost-effective approach to flexibly scale server capacity and dynamically allocate resources is to employ a server cluster. While a clustered approach is commonly used for designing scalable web servers [1, 40], clustered streaming media servers impose a different set of requirements. First, clustered web servers replicate content on all replicas, while doing so is prohibitive for clustered streaming servers due to the large sizes of media objects and the sheer sizes of video repositories. Thus, clustered media servers need to partition the content and workload among individual servers. Second, media sessions are long-lived, unlike web requests which have sub-second service times. Third, as argued above, media workloads are skewed, hard to predict a priori and have time-varying popularities.

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. Although DALA is designed as a clustered server architecture, it can also be employed as a clustered proxy in a content delivery network; streaming

CDNs face many of the same issues as servers—content with varying popularity with new content being cache on a daily basis. As a result, many of the key ideas in DALA can be applied directly for dynamically managing resources in CDN-based clustered streaming proxies.

Our DALA cluster consists of low cost commodity components that is easily configurable and tremendously flexible for updates – with respect to both content and server hardware. 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 content object based on its popularity—more resources are 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 present an analysis of the DALA approach and demonstrate its 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 describes the details of our DALA clustered server architecture. Section 3 presents an analytical model on evaluating the performance of DALA. Section 4 presents our experimental results. Section 5 discusses related work, and finally, Section 6 presents our conclusions.

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

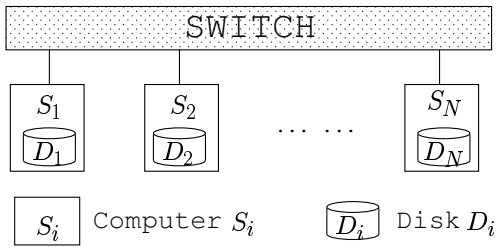

Figure 1: A Sample Cluster Server Infrastructure

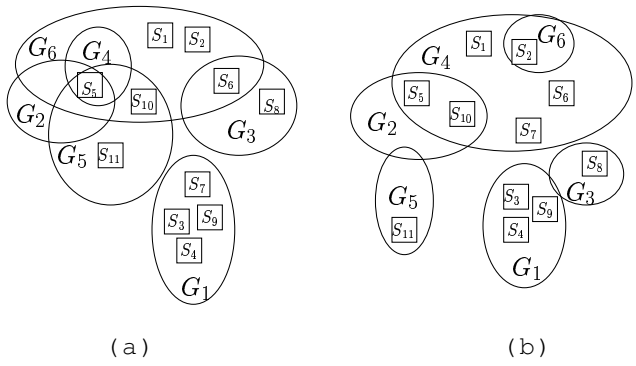

Figure 2: Conceptual DALA Server Cluster Design

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

$$t_j\text{: as the token holder for object } j$$

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 [22, 12, 49]; any such algorithm suffices for this purpose. Note that it is desirable to reserve a small fraction of bandwidth resource (e.g., that corresponds to one streaming session) for control messages such as for token passes. 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 (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. Further, if token passing is not too frequent, exchanging load information results in wasted messages. The advantage though is that a new

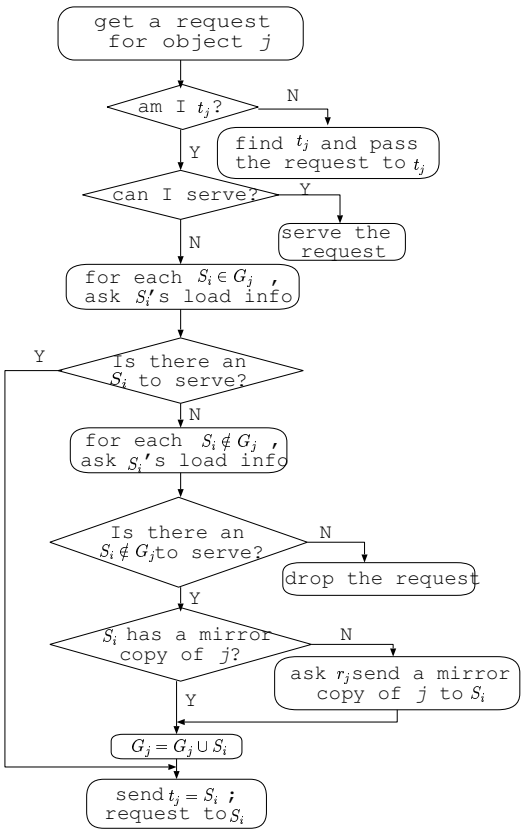

Figure 3: A Pictorial Representation of the DALA Demand Dissemination Algorithm

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 [37]. 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.

In addition, the token passing protocol used in our design is similar to other related standard algorithms. Thus, to handle failure situations, a variety of standard fault-tolerance techniques can be applied to our architecture [33]. For instance, if the token for an object is lost, the root server can assume the role of new token holder (similar to techniques employed in token-based LANs [30]). When the root server fails, a traditional election algorithm [48] can be applied to select a new root server from those servers who own mirror copies for the streaming object. Server failures change the membership of all groups to which a server belongs; such group membership changes have been studied in the context of reliable group communication and can be employed in our context as well [35].

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 (as a reasonably large initial part of the object should be in place before service starts in order to ensure continuous and smooth playback of the user). On the other hand, eager group expansion can result in wasted copying effort. A group expansion fails either when no candidate server is able to join, or when all existing members do not have the resource to replicate the object onto the new member over an extended period.

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 [21] 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 [40] and have exhibited substantial performance gains. Since web contents are served in best-effort manner, exploiting locality should be weighed against balancing the load on various servers. 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 consideration (so long as QoS requirements are not violated, the distribution of load among various server is of secondary importance). Given the long-lived nature of streaming sessions, we expect to observe significant performance gains using locality-aware request distribution policy.

3 An Analytical Model

In this section, we present a simple mathematical model for streaming media server clusters. Using the model, we study the improvement in the system performance by considering our demand adaptation and locality-aware request distribution techniques.

We consider a server cluster serving M movie clips. Each movie clip is of length T and is encoded at rate R bps. The cluster consists of N servers. Each server has L bytes of memory that is available for interval caching. The disk on each server has KR bps sustained bandwidth (data transferred average over disk seek time, rotational latency and data transfer time). For the simplicity of the model, we do not consider disk size and network bandwidth as constraints – these factors will be included in our simulation study in Section 4. We also do not consider server CPU as a bottleneck since we assume the cluster serves pre-encoded streaming content from disk, hence having no strong computation requirement. We assume the arrival of client requests is Poisson with rate λ, and the requests are uniformly distributed among the M movie clips. Furthermore, all requests sequentially access the media at the playback rate from the start to the end of the movie.

While demand adaptation allows under-utilized system resources associated with infrequently-requested movies to participate in servicing popular movies, and locality-aware request distribution increases the probability of requests being serviced directly from memory buffers, both approaches should improve the system performance. To quantify the effectiveness of these approaches, we compare the following four systems:

(i) Each request is uniformly distributed to one of the N servers and is rejected if it cannot be admitted by this server. Since an overloaded server in this system cannot utilize its peer server's remain capacity

and the request distribution does not consider previous request distribution, this system – non-adaptive non-locality-aware (NANL) – provides the baseline for our comparison.

(ii) Servers evenly partition the movie collection so that each server is responsible for M/N movies. A request is rejected if the corresponding server is overloaded. Since all requests of the same movie are directed to one server, this system inherently achieves locality-aware request distribution, however it is still non-adaptive (NALA).

(iii) Requests are uniformly distributed among N servers. If a request cannot be admitted on one server, it is forwarded to an under-utilized peer server unless all servers are fully utilized, in which case it is rejected. This system is adaptive but not locality-aware (DANL).

(iv) One token is created for each movie and is held by one of the servers. A token holder is responsible for all requests of the corresponding movie. When a server is fully loaded, new requests, along with the token for the movie being requested, are forwarded to an under-utilized peer server. A request is rejected only when all servers are fully utilized. This system is demand adaptive and locality-aware (DALA).

Note that the above systems do not assume disk capacity as a constraint, i.e., all movies can be available at each server, thus there is no cost to bring in a peer server to serve a request when adapting to demand.

For each of the above systems, we are interested in obtaining the request rejection rate D – the rate that requests are denied admission due to disk bandwidth constraint. To do that, we first derive the cache hit rate of a streaming media server that utilizes interval caching scheme.

3.1 Cache Hit Rate

In this subsection, we consider a single streaming media server that serves M' movies. Assuming client requests are Poisson with rate λ' (and no requested are rejected), the question of interest is: what is the fraction of requests that can be served from memory, i.e., the cache hit rate H, assuming interval caching policy? The interval caching cache management policy, in a nutshell, is to keep track of the content buffer between each pair of consecutive requests on the same streaming object, referred to as an *interval*, and store the intervals as basic caching units in memory. To maximize hit rate, caching preference is set such that smaller intervals are stored first till all allocated memory resource being used. Note that the length (size) of a cache interval remains constant during the playback of the two streaming requests that form the interval,

as they are regarding to the same streaming object. The implementation details on keeping track of interval buffers and smoothly handling new arrivals, departures or interval replacements can be found in [21].

We proceed by deriving the expression for each of the following parameters.

- Expected number of concurrent sessions:

  Since the requests for each movie are Poisson with rate λ'/M', and movie length is T, the number of concurrent sessions of each movie, C_i where $1 \leq i \leq M'$, is the number of requests (of the same movie) that arrive within time T. Thus,

  $$P[C_i = c] = \frac{(\lambda' T/M')^c e^{-\lambda' T/M'}}{c!}, c \geq 0$$

  The total number of concurrent sessions, C, is $\sum_{i=1}^{M'} C_i$ with expected value:

  $$E[C] = \sum_{i=1}^{M'} E[C_i] = \sum_{i=1}^{M'} \lambda' T/M' = \lambda' T$$

- Expected number of intervals formed by concurrent sessions:

  An interval is the time difference between two consecutive requests of the same movie being serviced. Thus, the number of intervals of movie i, W_i, is one less than the number of concurrent sessions of movie i, C_i where $1 \leq i \leq M'$. Thus,

  $$P[W_i = w] = \begin{cases} (1 + \lambda' T/M') e^{-\lambda' T/M'} & w = 0 \\ \frac{(\lambda' T/M')^{w+1} e^{-\lambda' T/M'}}{(w+1)!} & w \geq 1 \end{cases}$$

  Similarly, the expected number of total intervals $E[W]$, can be obtained as follows, where $W = \sum_{i=1}^{M'} W_i$:

  $$E[W] = \sum_{i=1}^{M'} E[W_i] = \sum_{i=1}^{M'} \sum_{w=1}^{\infty} \frac{w(\lambda' T/M')^{w+1} e^{-\lambda' T/M'}}{(w+1)!} = \lambda' T - M'(1 - e^{\lambda' T/M'})$$

- The number of intervals that can fit in cache when W intervals are formed by concurrent sessions

  Let X_i be the length of the i-th smallest interval, where $1 \leq i \leq W$. Considering interval caching scheme in which preference for memory caching is given to small intervals, the number of intervals that can fit in cache, V, is determined by

  $$V = \max\{v | \sum_{i=1}^{v} X_i R/8 \leq L\},$$

where the factor of 8 is introduced to convert the unit of bit to byte.

Without the condition that W intervals are formed by concurrent sessions, X_i would be the i-th order statistic of W exponentially distributed random variables (with mean M'/λ'), whose cumulative distribution function is

$$F_{X_i}(x) = (1 - e^{-\lambda' x/M'})^i e^{-\lambda' x(W-i)/M'}$$

However, since we know that the W intervals are formed by requests that arrive within T, the conditional distribution of the length of these W intervals is no longer exponential. Yet, in our model, we will still approximate them using exponentially distributed random variables. To match the expected value, we use the mean value $\frac{T}{W/M'+2}$ — there are $W/M'+1$ arrivals uniformly distributed within time T for each movie on average, breaking T into $W/M'+2$ pieces. The exponential approximation is good when the request rejection rate is low, as the underlying arrival process is indeed Poisson. However if the rejection rate is high, this approximation overestimates the coefficient of variation of the interval size distribution, producing a higher cache hit rate following the derivation steps next (we will see in the validation result that this approximation is close).

Using this approximation, we have

$$F_{X_i}(x) = (1 - e^{-(W/M'+2)x/T})^i e^{-(W/M'+2)x(W-i)/T}$$

And the expected value of X_i is

$$E[X_i] = \sum_{j=0}^{i-1} \frac{T}{(W-j)(W/M'+2)}$$

To obtain the average number of cached intervals, we use the following approximation:

$$\begin{aligned}
E[V] &\simeq \max\{v | \sum_{i=1}^{v} E[X_i] R/8 \le L\} \\
&\simeq \max\{v | \sum_{i=1}^{v} \sum_{j=0}^{i-1} \frac{T}{(E[W]-j)(E[W]/M'+2)} R/8 \le L\} \\
&= \max\{v | \sum_{i=0}^{v} \frac{(v-i)T}{(E[W]-i)(E[W]/M'+2)} R/8 \le L\} \\
&\simeq \max\{v | \int_{0}^{v} \frac{(v-x)T}{(E[W]-x)(E[W]/M'+2)} dx \le 8L/R\} \\
&= \max\{v | \int_{0}^{v} \frac{(v-x)T}{(E[W]-x)(E[W]/M'+2)} dx \le 8L/R\} \\
\Rightarrow \quad & E[V] + (E([W]-E[V])(\ln(E[W]-E[V]) - \ln(E[W])) = \frac{8(E[W]/M'+2)L}{RT}
\end{aligned}$$

- The mean cache hit rate

  The cache hit rate H with interval caching is the fraction of sessions served by cache over all concurrent sessions. We further approximate the mean cache hit rate $\bar{H}$ by $E[V]/E[C]$.

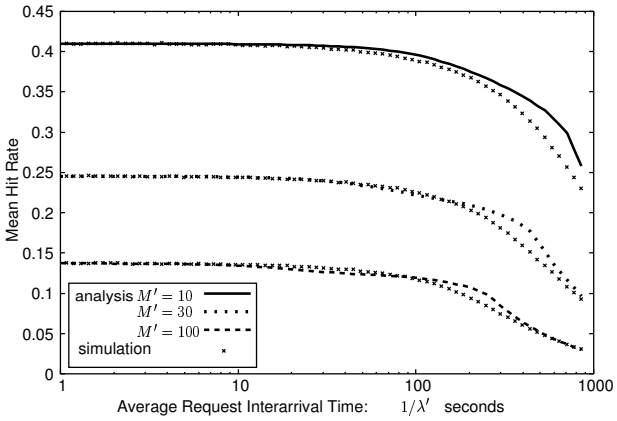

Figure 4: Mean cache hit rate versus mean request inter-arrival time

To validate our approximation, we compare the result of our analysis by using a simulation. The parameters that we use in the simulation are: movie encoding rate $R = 300$ Kbps, movie length $T = 5400$ seconds (90 minutes), and memory buffer size $L = 200$ MB. Figure 4 plots the average hit rate versus the mean request inter-arrival time $(1/\lambda')$ for $M' = 10, 30$ and 100 respectively. From Figure 4, we observe a close match between the simulation result and the analytical result.

We now can describe the average cache hit rate as a function of effective arrival rate λ' and the number of movies serviced: $\bar{H} = f(\lambda', M')$. Next, we evaluate the request rejection rate of an individual server.

3.2 Request Rejection Rate

In this subsection, we consider a single streaming media server system with Poisson request arrivals at rate λ''. Assuming the server's sustained disk bandwidth allows K concurrent streams being serviced from disk and the memory caching achieves an average hit rate $\bar{H}$ on the workload, we compute the mean request rejection rate $\bar{D}$.

If we assume the requests that cannot be serviced from memory caching are still Poisson (with rate

$(1-\bar{H})\lambda'')$, by applying Erlang-B formula, we obtain

$$\bar{D} \equiv g(\lambda'', K, \bar{H}) = \frac{((1-\bar{H})\lambda'' L)^K / K!}{\sum_{i=0}^{K} ((1-\bar{H})\lambda'' L)^i / i!} \tag{1}$$

where L is the length of the movies.

Since the mean cache hit rate $\bar{H}$ is a function of effective arrival rate $\lambda' = (1-\bar{D})\lambda''$ and the number of movies serviced M', i.e., $\bar{H} = f((1-\bar{D})\lambda'', M')$, substituting $\bar{H}$ in (1), we can solve for $\bar{D}$ numerically through an simple iterative procedure.

3.3 Effect of Demand Adaptation and Locality-aware Distribution

Now we apply our analysis to the four different systems described at the beginning of the section.

(i) NANL: $\bar{D} = g(\lambda/N, K, \bar{H})$ and $\bar{H} = f((1-\bar{D})\lambda/N, M)$

Since requests are uniformly distributed among servers, the request arrival rate to each server is λ/N and thus the effective arrival rate (for admitted requests) is $(1-\bar{D})\lambda/N$. The requests to each server are equal-likely to be for one of the M movies. Thus we can compute the mean cache hit rate by $\bar{H} = f((1-\bar{D})\lambda/N, M)$, and combined with $\bar{D} = g(\lambda/N, K, \bar{H})$, we can solve for the mean request rejection rate on one server, which is equivalent to that of the server cluster.

(ii) NALA: $\bar{D} = g(\lambda/N, K, \bar{H})$ and $\bar{H} = f((1-\bar{D})\lambda/N, M/N)$

Since each server service M/N movies and the request rate for each movie is λ/M, the request arrival rate to each server is thus λ/N. The above equations determine the mean rejection rate on any one server and thus on the server cluster.

(iii) DANL: $\bar{D} = g(\lambda, NK, \bar{H})$ and $\bar{H} = f((1-\bar{D})\lambda/N, M)$

Since the cluster rejects request only when all servers are fully utilized, we can consider the system as a single server with disk capacity of NK concurrent sessions when computing the mean request rejection rate: $\bar{D} = g(\lambda, NK, \bar{H})$. Note that memory caches are distributively maintained by each server, thus the mean cache hit rate is still determined by $\bar{H} = f((1-\bar{D})\lambda/N, M)$.

(iv) DALA: $\bar{D} = g(\lambda, NK, \bar{H})$ and $\bar{H} = f((1-\bar{D})\lambda/N, M/N)$

Since no request is rejected until all servers are fully utilized, we can use $\bar{D} = g(\lambda, NK, \bar{H})$ to compute the mean request rejection (same as DANL). The token scheme ensures that consecutive requests regarding to the same movie are forwarded to the same server. If we assume tokens are evenly

distributed among servers when system is in steady state, then each server should be responsible for M/N movies at one time and the total request arrival rate to each server should be λ/N. Thus, we use $\bar{H} = f((1-\bar{D})\lambda/N, M/N)$ to capture the memory caching behavior.

Note that in the above, we have focused on any one of the servers in the cluster for the expected system performance. Since random (in uniform) split of Poisson arrivals remains Poisson, the base assumption of the single server system analysis in the previous subsections still applies, whereas we leave the validation of the model for the cluster to the comprehensive simulation study in Section 4.

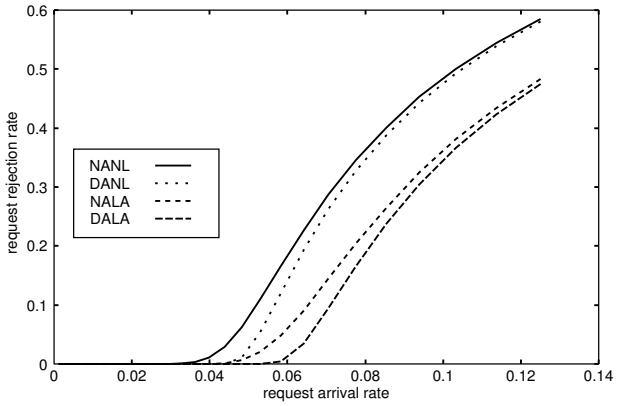

Figure 5: Request rejection rate versus request arrival rate

Figure 5 shows the request rejection rate as the function of the request arrival rate for the four systems. The parameters are $M = 100$, $N = 5$, $L = 200M$, $K = 50$, $T = 5400s$ and $R = 300Kbps$. From Figure 5, we observe that all four systems have virtually zero request rejection when request arrival rate is low; as request rate increases above a certain level, the request rejection rate starts to climb. Compared to NANL, both DANL and NALA can support a much higher ($\simeq 30\%$ larger) request rate before rejecting requests. As request rate increases, the advantage of DANL over NANL decreases. Meanwhile the differences between DANL and NALA increases. This is because, at a high request arrival rate, all system resources can be fully utilized, leaving little room for improvement by demand-adaptation. On the other hand, utilizing memory caching more effectively significantly increases system capacity. Therefore NALA produces a lower request rejection rate in comparison to DANL. Furthermore, DALA architecture achieves the highest request rate ($\simeq 60\%$ more than that of NANL) before denying requests, demonstrating the combined performance enhancement due to both locality-aware request distribution and demand adaptive resource allocation.

4 Experimental Results

Our model in Section 3 has evaluated the system performance of DALA architecture in terms of its stationary behavior when system is in steady state. We now evaluate the system behavior when the request demand is dynamic, i.e., the popularity of streaming objects changes over time. We do so through simulation. In our simulation, we have implemented the protocol as described in Section 2 with the lazy options. We include detailed factors that are ignored in the analytical model, such as the constraints due to network bandwidth and disk space, and system overhead caused by replicating objects to peer servers. To demonstrate the general applicability of our design and analysis, we purposely vary the system settings such as the encoding rate, the data repository size, the number of servers in the cluster, and the disk and network bandwidth. We next present our experimental settings and the corresponding evaluation results.

4.1 Effect of Demand Adaptation

Our first experiment simulates a flash crowd 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 6 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.

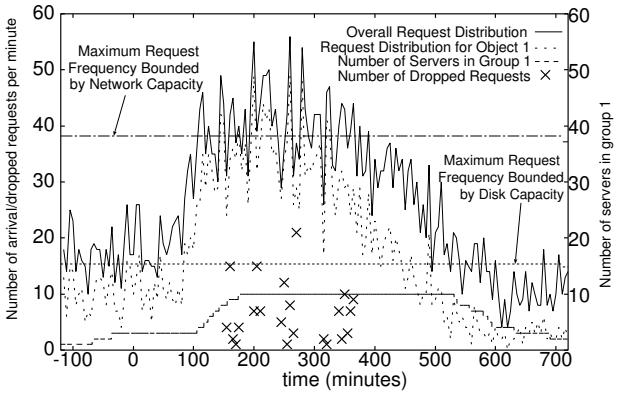

Figure 6: Effect of dynamically changing workloads.

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 the DALA adapts to changing workload and makes judicious use of cluster resources.

4.2 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.

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

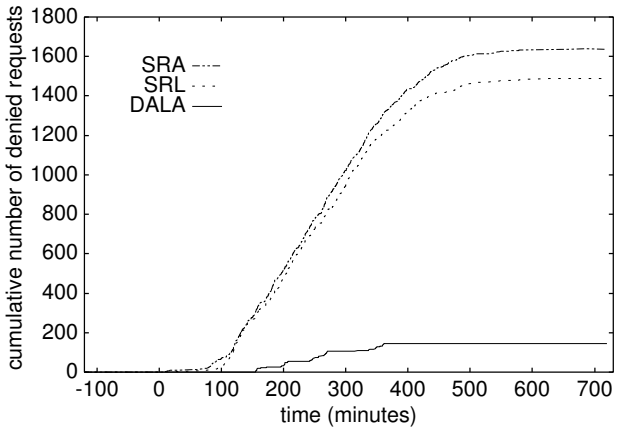

Figure 7: Comparing Different Request Distribution Policies

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 7 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.

4.3 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

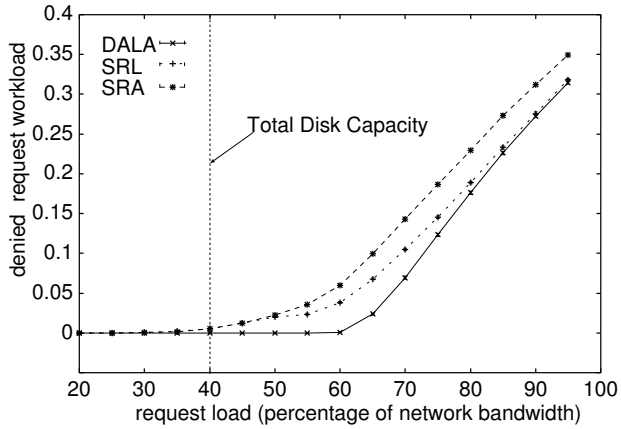

Figure 8: Comparing the Capacity of Server Clusters

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 8 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 percentile confidence interval of the dropped request load ratio. We are actually showing the confidence intervals in Figure 8. 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 8, 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

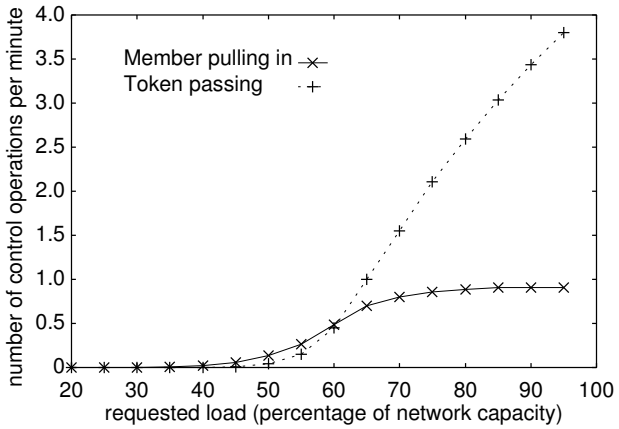

Figure 9: Control Overhead of DALA

be increased by exploiting request locality. Note that this result echos the analytical result from Section 3, evidenced by the high similarity between the shape of the curves in Figure 8 and in Figure 5. This suggests suggests that the model in Section 3.3 captures the essence of the system very well.

Finally, Figure 9 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. Same as when studying the dropped workload, we also calculate the mean and 95 percentile confidence interval per 100 sample runs in quantifying the system overhead, and we show the confidence intervals in Figure 9 as well. We observe a very low control overhead both in token passing and group expansions. Although shown in the same graph, the two control operations incur different level of system overhead. Token passing is relatively light-weighted, involving exchanges of signaling messages. Group expansions on the other hand can be heavy-weighted — unless a copy of the streaming object already exists on the newly included group member (due to lazy replacement policy), one group expansion induces one object replication. Depending on a design tradeoff on how much content (the fraction of the streaming object) has to be in place before new group member starts service, such object replication may increase to different extent the playback initiation latency experience by the end user. Furthermore, the added replication would take system resource that could be used to serve user request instead. However, as we will see in the next subsection, the dynamic adjustment of group memberships, while reducing system capacity with additional replications, in fact improves the effective service capacity of the system.

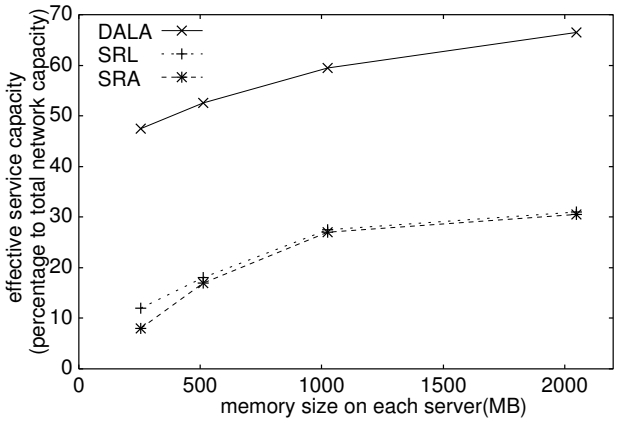

Figure 10: Effective service capacity of the server cluster

4.4 Effective Service Capacity

In many circumstances, we are interested in the maximum workload the system can support without having to drop request. 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 effective service capacity, in our simulation, we gradually increase the request arrival rate, thus increase the workload, by 0.5% each time from 0 to 100% comparing to the network bandwidth. On each workload level, we conduct 30 experiments and test the hypothesis that the denied workload is zero. If the denied workload is significantly different than zero (reject the hypothesis with 95 percentile confidence), 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 4.3 and also use the same server cluster configuration except that we vary the size of memory on each server. Figure 10 compares the effective service capacity of different architectures when memory sizes on each server are 256 MB, 512 MB, 1 GB and 2 GB respectively. In all cases, DALA shows a significant higher effective service capacity than SRL or SRA. However, SRL and SRA do not show much differences on effective service capacity, especially when memory size is sufficiently large. This can also be inferred from Figure 8 since exploiting memory caching mainly improves the system performance on high-load or overload situation. For all mechanisms, the effective service capacity increases sub-linearly with more memory added on each server.

Figure 11 shows another set of experiments where each server in the cluster has 1GB memory while

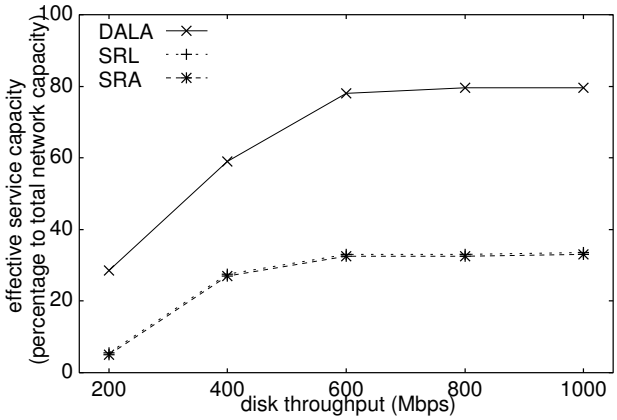

Figure 11: Effective service capacity of the server cluster

we change the disk throughput on each server. Consistent with previous experiment, DALA shows 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. The system can benefit from using a better disk with higher throughput. Once the disk throughput reaches 600Mbps (comparing to 1Gbps of network bandwidth), due to effects of memory caching, improve disks throughput can no longer help the system achieving higher effective capacity. Note that even when disk throughput is as high as network bandwidth, the effective service capacity of DALA does not reach 100% of network capacity. This is due to two factors: (a) dynamic replication itself consumes system resource in achieving better load-balancing among servers of different groups (b) bursty request arrivals, captured by the Poisson arrival process, lead to temporary system overload and request rejections even when the sustained workload is less than system capacity.

5 Related Work

Today's media delivery architectures consist of three tiers: server, proxy and end-client. Not surprisingly, researchers have focused on scaling system capacity by designing adaptive techniques at all three tiers. At the server tier, research has focused on design of distributed, clustered servers. At the proxy tier, researchers have proposed clustered proxies and content delivery systems, while at the client tier, peer-to-peer architectures have been proposed as a method for scaling system capacity and handling workload dynamics. Our related work is grouped into these categories, which we review below.

Distributed and Clustered Servers: Early work on video on demand systems focused primarily on single node servers. This work targeted a variety of topics such as admission control [22, 12], scheduling [49], striping [47, 31, 28] and caching [21].

Since the capacity of a single server is limited despite these optimizations, researchers began focusing on distributed and clustered server design [8, 45, 6, 32, 14, 51]. Some representative examples include the Tiger server from Microsoft [8], Exedra from U. Toronto [6], Rio from UCLA [32], FlexSplit from HP [14] and Yima from USC [46].

The Tiger distributed video server employs network striping across a cluster of nodes and uses a centralized controller to coordinate these nodes [8, 9]. 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 [6, 5, 7]. 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.

The FlexSplit architecture [13, 52].assumes media content is fully replicated by all servers and designs a request distribution and server load balancing strategy such that requests for different parts of a media stream are directed to different server nodes. The split points are dynamically determined by keeping track of the histogram of request distribution. DALA however divides requests in the temporal domain by utilizing a token passing scheme, which fully exploits the locality of demands. Another key advantage of DALA over the above systems is its easiness for configuration – new servers or new media contents can be added into an active server cluster directly with almost no configuration overhead or disturbance to the on-going services. This is particularly important in realistic server systems where content updates are frequent and requests toward newly introduced files are found dominant [13].

The RIO server [32, 31] employs a novel randomized data allocation strategy to alleviate hotspots in the storage system; they show via analysis that such a strategy is superior to striping-based placement in terms of spreading the workload across multiple storage entities. DALA employs a dynamic replication as a strategy for achieving the same objective.

Dynamic replication strategies have also been studied by other researchers [17, 16]. These studies have shown that dynamically varying the degree of striping and the degree of replication can significantly enhance the ability of a server to deal with load hotspots. DALA directly builds on these results via its ability to dynamically replicate content based on user demand.

Yima is a second generation streaming server from USC that employs a fully distributed architecture with multiple nodes [46, 53]. Yima has no single point of failure, allows disks to be added or removed without system disruption, and employs sophisticated buffering flow-control mechanisms for VBR media. At an architecture level, DALA has similarities with Yima since both employ multiple server nodes and have fail-over strategies. The specific focus of this paper, however, is on dynamic replication and exploiting locality to handle time-varying workloads.

A decentralized algorithm that allocates server resources to replicated servers was studied in [34]. The focus of the work is on reducing the network distance between each client and a nearby replicated server that has the requested content. In this sense, the effort is related to work on content delivery networks where content is placed on proxies that are closest to end-clients that request such content. Since DALA nodes are part of a server cluster, all server nodes are at the same distance from a particular end-client; thus, such optimizations are not directly applicable to server clusters such as DALA.

Clustered Proxies and Content Distribution Networks: A complementary technique to scale system capacity is to employ streaming proxies or a streaming content distribution network. Proxies cache content and can deliver popular content from the cache and alleviate server load. By dynamically varying the set of cached objects, proxies or CDNs can adapt to varying workloads. Consequently, there have been several studies on on distributed caching of streaming media content in the Internet [11, 20].

In SILO [11], 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 SILO require long-term popularity statistics, which is not a requirement in DALA.

A Spectrum architecture that partitions distributed content management system into three functional components, namely content manager, policy manager and storage manager has also been proposed [20] . The focus of this work is on providing flexibility for content sharing as opposed to on performance optimization as studied in DALA.

A cluster-based streaming proxy that uses popularity statistics to dynamically decide what set of objects

to cache and on which node of the cluster has also been proposed [26, 27]. Such a cache replacement policy in a distributed cluster of proxies has similar goals to our DALA work; however, DALA can additionally employ locality-aware techniques to further optimize system performance.

Middleman is a cooperative set of video proxy servers that coordinate with one another to service streaming requests [2]. Cooperative caching of streaming content is a key focus of Middleman, where requests received by one proxy are handed over to another one that caches the requested content. Such handoffs are similar to the token passing techniques employed by DALA. However, DALA additionally employs dynamic replication techniques to scale system capacity.

Mocha is a cluster of proxies that employ replication and caching to improve the quality of content seen by end-users [42, 43], They employ fine-grain prefetching and fine-grain replacement to deliver layered content to end-users. Such techniques for replicating and scaling layered video are complementary to dynamic replication in DALA.

In the context of scalable network delivery, 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 [4, 41, 29, 19, 10]. These techniques complement our work, since all of them can be employed by DALA as well.

Peer-to-peer streaming: Recently several efforts have focused on leveraging end-client resources to scale system capacity. Such systems are based on a peer-to-peer architecture where clients can serve locally stored content to other peer clients and thereby alleviate load on servers and proxies. They can also mask server or network failures, since a peer can continue to stream content even when an origin server becomes available. Such P2P systems can be adaptive, since the initial set of clients that fetch a new popular video clip can immediately start serving it to other clients. Peer-to-peer streaming is an active area of ongoing research and we only present a brief overview of recent work.

Cooperative Networking (CoopNet) project from Microsoft investigates how end-peers can improve the performance of end-servers [39, 38]. This is a hybrid approach where cooperation from peers complements traditional client-server communication rather than replacing it. The basic idea is to leverage client resources to alleviate the impact of sudden flash-crowds. In DALA, servers cooperate with one another to dynamically replicate content and handle workload spikes; further, DALA can fully leverage ideas from CoopNet to also employ end-clients to serve popular content and reduce load on servers. Since CoopNet is inherently designed to supplement servers, both DALA and CoopNet architectures can coexist and leverage one another's

dynamic workload handling capabilities.

Placement and dynamic replication of streaming objects in wireless peer-to-peer networks has been studied in [23, 25]. The C2P2 system replicates streaming content dynamically in a P2P system of mobile vehicular nodes [24]. Dynamic replication and placement on peer nodes is similar to dynamic caching, where new content is fetched or replicated as it becomes popular and less popular old content is removed. DALA also employs dynamic replication and deletion of old replicas, although on a cluster of server nodes as opposed to end-clients. Second, these efforts have focus on wireless environments or on handling node mobility, neither of which is an issue in DALA.

Finally, mesh-based P2P streaming and the use of receiver-driven overlays for P2P streaming have been studied in [36, 44]. The focus of these efforts is on network topologies for delivering content in a peer-to-peer fashion and is complementary to the issues studied in DALA. Altruism issues in peer-to-peer streaming systems has also been studied [18]; these issues are also orthogonal to the ones considered in DALA.

At a high-level, both peer-to-peer systems and CDN proxies complement a server architecture; both can absorb some of the client workload but do not completely eliminate load at the server. Thus, a server architecture still needs to be scalable and adaptive to respond to dynamic workloads. Finally, as indicated earlier, a DALA cluster can be employed as a proxy cluster in a CDN to handle workload dynamics, when enhanced with suitable cache replacement policies.

6 Conclusions

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. We developed both analytical models and simulations to examine the performance of DALA. Our evaluation and 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 our future work, we would like to extend DALA to a distributed server architecture over the Internet.

References

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

[2] Soam Acharya and Brian Smith. MiddleMan: A Video Caching Proxy Server. In *Proceedings of NOSSDAV 2000, Chapel Hill, NC, USA*, June 2000.

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

[4] 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.

[5] S V. Anastasiadis, K. Sevcik, and M. Stumm. Disk-striping scalability in the exedra media server. In *Proceedings of ACM/SPIE Multimedia Computing and networking 2001*, pages 175–189, January 2001.

[6] 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.

[7] Stergios V. Anastasiadis, Kenneth C. Sevcik, and Michael Stumm. Scalable and fault-tolerant support for variable bit-rate data in the Exedra streaming server. *ACM Transactions on Storage*, 1(4):419–456, November 2005.

[8] 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.

[9] W J. Bolosky, R P. Fitzgerald, and J R. Douceur. Distributed schedule management in the tiger video fileserver. In *Proceedings of the sixteenth ACM symposium on Operating Systems Principles (SOSP'97), Saint-Malo, France*, pages 212–223, December 1997.

[10] M. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley. Periodic broadcast and patching services: Implementation, measurement and analysis in an internet streaming media testbed. In *Proceedings of the Ninth ACM Conference on Multimedia, Ottawa, Canada*, October 2001.

[11] 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. *IEEE Journal on Selected Areas in Communications (JSAC)*, September 2002.

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

[13] L. Cherkasova and M. Gupta. Analysis of enterprise media server workloads: Access patterns, locality, content evolution, and rates of change. *ACM/IEEE J. Transactions on Networking, (TON)*, 12:781–794, Oct. 2004.

[14] L. Cherkasova and W. Tang. Sizing the streaming media cluster solution for a given workload. http://www.hpl.hp.com/techreports/2004/HPL-2004-65.html.

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

[16] C.F. Chou, L. Golubchik, and J.C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS), San Francisco, CA*, August 2000.

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

[18] Y. Chu and H. Zhang. Considering altruism in peer-to-peer internet streaming broadcast. In *Proceedings of ACM NOSSDAV*, 2004.

[19] 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.

[20] Chuck Cranor, Rick Ethington, Amit Sehgal, David Shur, Cormac Sreenan, and Kobus van der Merwe. Design and implementation of a distributed content management system. In *NOSSDAV*, 2003.

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

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

[23] S. Ghandeharizadeh and T. Helmi. An evaluation of alternative continuous media replication techniques in wireless peer-to-peer networks. In *Proc. of Third International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), San Diego, CA*, September 2003.

[24] S. Ghandeharizadeh and B. Krishnamachari. C2p2:a peer-to-peer network for on-demand automobile information services. In *Proceedings of the First International Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE'04), Zaragoza, Spain*, 2004.

[25] S. Ghandeharizadeh, B. Krishnamachari, and S. Song. Placement of continuous media in wireless peer-to-peer networks. *IEEE Transactions on Multimedia*, 6(4), April 2004.

[26] Y. Guo, Z. Ge, B. Urgaonkar, P. Shenoy, and D. Towsley. Dynamic cache reconfiguration strategies for cluster-based streaming proxies. In *Proceedings of the Eighth International Workshop on Web Content Caching and Distribution (WCW 2003), Hawthorne, NY*, September 2003.

[27] Y. Guo, Z. Ge, B. Urgaonkar, P. Shenoy, and D. Towsley. Dynamic cache reconfiguration strategies for cluster-based streaming proxies. *Computer Communications*, 29(10):1710–1721, June 2006.

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

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

[30] J. Kurose and K. Ross. *Computer Networking: A Top Down Approach Featuring the Internet*. Addison Wesley Longman, 2001.

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

[32] 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.

[33] K. Birman. *Reliable Distributed Systems, Technologies, Web Services, and Applications*. Springer Verlag, 2005.

[34] B. Ko and D. Rubenstein. Distributed server replication in large scale networks. In *Proceedings of the 14th ACM International Workshop on Network and operating systems support for digital audio and video (NOSSDAV) Cork, Ireland*, pages 127–132, 2004.

[35] K.P.Birman. The process group approach to reliable distributed computing. *Communications of the ACM (Commun.ACM)*, 36(12):36–53, 1993.

[36] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of IEEE INFOCOM, Anchorage, Alaska*, May 2007.

[37] Michael Mitzenmacher, Andra 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.

[38] V. N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA*, March 2002.

[39] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proc. of ACM NOSSDAV, Miami Beach, FL*, May 2002.

[40] 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.

[41] Jehan-Franois Pris, 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.

[42] R. Rejaie, H. Yu, M. Handely, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet, March 2000.

[43] Reza Rejaie and Jussi Kangasharju. Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming. In *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'01), Port Jefferson, NY, USA*, pages 3–10, June 2001.

[44] Reza Rejaie and Shad Stafford. A framework for architecting peer-to-peer receiver-driven overlays. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video Cork, Ireland*, June 2004.

[45] 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.

[46] C. Shahabi, R. Zimmermann, K. Fu, and S D. Yao. Yima: A second generation continuous media server. *IEEE Computer*, pages 56–64, June 2002.

[47] 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.

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

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

[50] Youtube bandwidth: Terrabytes per day. `http://blog.forret.com/2006/05/youtube-bandwidth-terabytes-per-day/`.

[51] Q. Zhang, L. Cherkasova, and E. Smirni. Flexsplit: A workload-aware, adaptive load balancing strategy for media clusters. In *Multimedia Computing and Networking.(MMCN'2006)*, San-Jose, CA, Jan. 2006.

[52] Q. Zhang, L. Cherkasova, and E. Smirni. Flexsplit: A workload-aware, adaptive load balancing strategy for media clusters. In *Proc. of Multimedia Computing and Networking.(MMCN'2006), San-Jose, CA*, 2006.

[53] R. Zimmermann, K. Fu, C. Shahabi, D. Yao, and H. Zhu. Yima: Design and evaluation of a streaming media system for residential broadband services. In *Proceedings of the VLDB 2001 Workshop on Databases in Telecommunications (DBTel 2001), Rome, Italy*, September 2001.