FlexSplit: A Workload-Aware, Adaptive Load Balancing Strategy for Media Cluster

Qi Zhang
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
qizhang@cs.wm.edu

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303
lucy.cherkasova@hp.com

Evgenia Smirni
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
esmirni@cs.wm.edu

Abstract *A number of technology and workload trends motivate us to consider a new request distribution and load balancing strategy for streaming media cluster. First, in emerging media workloads, a significant portion of the content is short and encoded at low bit rates. Additionally, media workloads display a strong temporal and spatial locality. This makes modern servers with gigabytes of main memory well suited to deliver a large fraction of accesses to popular files from memory. Second, a specific characteristic of streaming media workloads is that many clients do not finish playing an entire media file that reflects the browsing nature of a large fraction of client accesses. In this paper, we design and evaluate two novel load-balancing strategies for media server cluster: FlexSplit and FlexSplitLard, that aim to efficiently utilize the combined cluster memory by exploiting specific media workload properties. New strategies "tune" their behavior to reflect media file popularity changes and other dynamics exhibited by media workload over time. Adaptive nature and improved cluster performance make these strategies an attractive choice for handling dynamically changing workloads by media server cluster.*

1 Introduction

Streaming high quality audio and video from a central server complex to a large number of clients is a challenging and resource intensive task. Transmitting media files requires more computing power, bandwidth and storage, and it is more sensitive to network jitter than web objects.

Media server clusters are used to create scalable and highly available solutions. While the network bandwidth usage by media applications can be considered as a primary component in the service billing, the cost to manage, operate, and power more traditional resources like CPU, memory, and storage should be taken into account as the additional important targets of resource management and allocation.

Traditional load balancing solution for a media server cluster, such as *Round-Robin (RR)*, tries to distribute the requests uniformly to all the machines. However, this may adversely affect an efficient memory usage because the frequently accessed content is replicated (cached) across the memories of all the machines. Earlier analysis [3, 8] shows that emerging media workloads exhibit a high degree of temporal and spatial reference locality (i.e. a high percentage of requests are accessing a small subset of media files) and exhibits a strong sharing pattern (i.e. accesses to the same file come in "bursts"). Typically, a media server throughput is significantly higher (3-10 times) when media streams are delivered from memory versus from disk. A significant portion of media content is represented by short and medium videos (2 min-15 min), and the encoding bit rates, targeting the current Internet population, are typically 56 Kb/s - 256 Kb/s CBR. These popular streaming objects have footprints on the order of 10 MB. At the same time, modern servers have up to 4 GB of main memory, meaning that a large fraction of the accesses to popular media objects can be served from memory, even when a media server relies on traditional file system and memory support and does not have additional application level caching. Thus, the locality available in a particular workload will have a significant impact on the behavior of the system because serving content from memory will incur much lower overhead than serving the same content from disk. This observation led researchers to a design of "locality-aware" balancing strategies like *LARD* [9] and *DALA* [6], which aim to avoid the unnecessary file replication (caching) by different cluster nodes to improve the overall system performance.

The other specific characteristic of streaming media workloads is that a significant amount of clients do not finish playing an entire media file [2, 3]. Typically, this reflects the browsing nature of client accesses, client time constraints, or QoS-related issues. Most incomplete sessions (i.e. terminated by clients before the video is finished entirely) access only the initial segments of media files. In two streaming media workloads analyzed in [3], only 29% of the accesses in the first workload and 12.6% of the accesses in the second workload finish the playback of the files. 50%-60% of the accesses in both workloads last less than 2 minutes. This high percentage of incomplete accesses as well as a high number of sessions accessing only the initial part of the file create a very special resource usage model, which is widely considered for streaming media cache design [10].

In this paper, we design and evaluate two novel load-balancing strategies for media server cluster: *FlexSplit* and *FlexSplitLard*, that aim to efficiently utilize the combined cluster memory by exploiting media workload properties discussed above. Under the new strategies, a media file is split into a sequence of continuous, different size file pieces for streaming by different nodes in the cluster, and thus, it is delivered to the client by coordinated ensemble of nodes in media cluster. The file "split" points (i.e. the "cut-off" points in the file that define what portions of a media file are served by different cluster nodes) are dynamically updated over time. This feature enables the proposed strategies to deal with media file popularity changes and other dynamics exhibited by media workload over time.

We evaluate the efficiency of the new strategies using a simulation model and a synthetic workload closely imitating parameters of real media server workload. Our evaluation of *FlexSplit* and *FlexSplitLard* compared to the traditional *RR* load balancing strategy reveals a factor of ten improvement in the number of dropped sessions and 15% higher available capacity in media cluster over time. Our evaluation of *FlexSplit* and *FlexSplitLard* compared to *LARD* strategy reveals a factor of two improvement in the number of dropped streams as well as 4% higher available capacity in media cluster on average. Additionally, new strategies support much more predictable resource usage pattern compared to LARD that makes them an attractive choice for handling dynamically changing workloads by media server cluster. The remainder of the paper presents our results in more detail.

2 Media Server Capacity and Segment-Based Memory Model

Commercial media servers are characterized by the number of concurrent streams that can be supported by a server without loosing a stream quality, i.e. while meeting the real-time constraint of each stream. A set of simple basic benchmarks were introduced in [4] that can establish the scaling rules for server capacity when multiple media streams are encoded at different bit rates:

- *Single File Benchmark* measuring a media server capacity when all the clients in the test are accessing the same file, and
- *Unique Files Benchmark* measuring a media server capacity when each client in the test is accessing a different file.

The measurement results in [4] show that the scaling rules for server capacity are non-linear when multiple media streams are encoded at different bit rates and that the media server performance is much higher (for some disk/file subsystems up to 10 times higher) when the media requests are served from memory versus from disk. Using the benchmark measurements, one can derive a cost function which defines a *fraction* of system resources needed to support a particular media stream depending on the stream bit rate and type of access (memory file access or disk file access)as derived in [5].

- $cost^{disk}_{X_i}$ - value of the cost function for a stream with disk access to a file encoded at X_i Kb/s.
- $cost^{memory}_{X_i}$ - value of the cost function for a stream with memory access to a file encoded at X_i Kb/s.

Let W be the current workload processed by a media server, where

- $X_w = X_1, ...X_{k_w}$ - a set of distinct encoding bit rates of the files appearing in W,
- $N^{memory}_{X_{w_i}}$- a number of streams having a memory access type for a subset of files encoded at X_{w_i} Kb/s,
- $N^{disk}_{X_{w_i}}$ - a number of streams having a disk access type for a subset of files encoded at X_{w_i} Kb/s.

Then the service demand to a media server under workload W can be computed by the following capacity equation:

$$Demand = \sum_{i=1}^{k_w} N^{memory}_{X_{w_i}} \times cost^{memory}_{X_{w_i}} + \sum_{i=1}^{k_w} N^{disk}_{X_{w_i}} \times cost^{disk}_{X_{w_i}}. \tag{1}$$

We use the *capacity equation* (1) to evaluate the capacity requirements of media workload on a particular media server configuration.

In order to assign a *cost* to a new media request, we need to evaluate whether a request will be streaming data from memory or will be accessing data from disk. Note, that memory access does not assume or require that the whole file resides in memory – if there is a sequence of accesses to the same file, issued closely in time to one another, then the first access may read a file from disk, while the subsequent requests may be accessing the corresponding file prefix from memory. For this classification, we use a *segment-based memory model* proposed in [5] that reflects data stored in memory as a result of media file accesses. The basic idea of computing the request access type exploits the real-time nature of streaming media applications and the sequential access to file content. Let $Size^{mem}$ be the size of memory in bytes [1]. For each request r, we have information about the media file requested by r, the duration of r in seconds, the encoding bit rate of the media file requested by r, the time t when a stream corresponding to request r is started (we use $r(t)$ to reflect it), and also for earlier finished requests, we have the time when a stream initiated by request r is terminated.

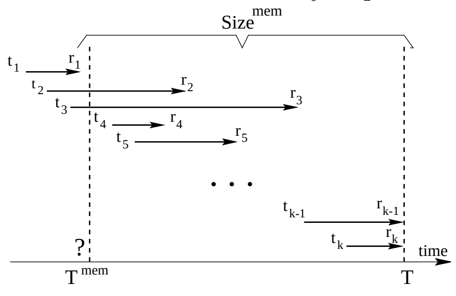

Figure 1: Memory state computation example.

Let $r_1(t_1), r_2(t_2), ..., r_k(t_k)$ be a recorded sequence of requests to a media server. Given the current time T and request $r(T)$ to media file f, we compute some past time T^{mem} such that the sum of the bytes stored in memory between T^{mem} and T is equal to $Size^{mem}$ as shown in Figure 1. This way, the files' segments streamed by the media server between times T^{mem} and T will be in memory. Thus, we can identify whether request r will stream file f (or some portion of it) from memory.

3 Load Balancing Solutions for Media Server Clusters

Media server clusters are used to create scalable and highly available solutions. We assume that each media server in a cluster has access to all the media content. Therefore, any server can satisfy any client request. A cluster of N nodes represents N times greater processing power, and at the same time, it has N times larger combined memory. Thus, for an accurate sizing of a cluster solution, we need to take into account both: its increased processing power and its increased memory size.

Traditional load balancing solution for a media server cluster, such as *Round-Robin (RR)*, tries to distribute the requests uniformly to all the machines. However, this may adversely affect an efficient memory usage because the frequently accessed content is replicated across the memories of all the machines. With this approach, a cluster having N times bigger memory (which is the combined memory of N nodes) might effectively utilize almost the same memory as one node. This observation led researchers to a design of "locality-aware" balancing strategies like *LARD* [9] and *DALA* [6], which aim to avoid the unnecessary file replication to improve the overall system performance. Although LARD can obtain good memory hit ratios, it does not balance the load in the cluster equally since it ignores the characters of transient workload and file popularity. Recently some policies, such as *AdaptLoad* [12] are proposed to achieve both "workload-aware" and "locality-aware" behavior. AdaptLoad monitors and predicts the incoming workload and balances the load using the empirical size distribution of the requested documents. It dispatches a request based on its requested file size. However AdaptLoad can not be directly applied to streaming media scenario. Since many clients (as high as 80% of all) do not finish playing the whole video, the distribution of durations of media sessions (i.e. client accesses) and the distribution of original media file durations can be vastly different. Intuitively, we need to modify AdaptLoad and its heuristics to effectively deal with incoming incomplete client media sessions.

[1]Here, the memory size means an estimate of what the system may use for a file buffer cache.

Our load balancing strategy FlexSplit is inspired by AdaptLoad, but instead of considering the file sizes, we focus on the prefix duration of the requests and aim to achieve high memory hit ratios for the early play durations of the streaming workload as well as equally load balancing in the cluster. In the next section, we will describe FlexSplit strategy in more detail.

I. FlexSplit strategy

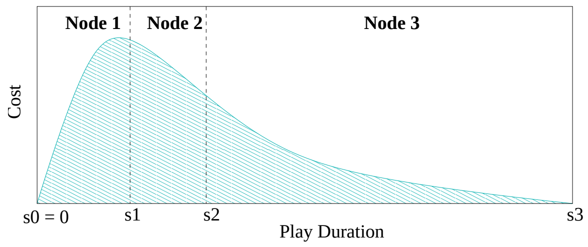

Figure 2: Illustration of histogram of costs by play duration for $N = 3$.

Assume that there are N identical nodes in the cluster, and M multimedia files stored in the system. The total *cost* of a stream r with play duration D_r and encoding bitrate X_r will be

$$cost_r = D_r P_r cost_{X_r}^{memory} + D_r(1 - P_r) cost_{X_r}^{disk}, \tag{2}$$

where P_r is the percentage of its duration under memory hit. FlexSplit load balancing policy focuses on partition the total costs of all streaming requests into N nodes equally. To this end, the entire duration is partition into N disjoint intervals, $[s_0 \equiv 0, s_1), [s_1, s_2)$, up to $[s_{N-1}, s_N \equiv \infty)$, so that node i, for $1 \le i \le N$, is responsible for satisfying requests whose current streaming duration is between s_{i-1} and s_i seconds. As illustrated in Figure 2, each interval $[s_{i-1}, s_i)$ should be set so that the requests routed to server i contribute a fraction $1/N$ to the value of the expected total costs, $\overline{S}$, that is, for $1 \le i \le N$,

$$\sum_{x=s_{i-1}}^{s_i-1} F(x) \approx \frac{1}{N}\sum_{0}^{\infty} F(x) = \frac{\overline{S}}{N}, \tag{3}$$

where $F(x)$ is the total costs for the x^{th} second play duration.

It is hard to predict if a request will be streaming a file from memory or disk at the time it arrives the system, and what portion of the file will be streamed from memory, because the request might generate more sub-requests to different nodes sequentially. Therefore we simply give "higher" cost to tails of the files given that tails of the files are not frequently requested and have higher probability to have memory misses. We compute the average play duration $access_i$ of all the previous streams for file i. Then for all the play seconds after $access_i$, they will have a higher cost by multiplying a scalar *weight*. The value *weight* is important for the performance. If $weight = 1$, then we totally ignore the high cost of disk streams. In this way, the first several servers will be under-loaded and the last several servers will be over-loaded because of the lower memory hit ratios of the last servers. On the other hand, if $weight = cost_{X_i}^{disk}/cost_{X_i}^{memory}$, the first servers will be over-loaded because we over-estimate the costs in the last servers by considering all the streams in the last servers as disk streams. Therefore *weight* should be a value in the range $(1, cost_{X_i}^{disk}/cost_{X_i}^{memory})$, and it also depends on the hit ratio and the cost function in each server. Currently *weight* is decided by off-line searching for the optimal dropping ratios, but it is also possible to develop an adaptive on-line policy.

We employ a monitor window, i.e., every K departures (10,000 in our experiments), to monitor and update the split points. The splitting interval $[s_0 \equiv 0, s_1), [s_1, s_2)$, up to $[s_{N-1}, s_N \equiv \infty)$ are updated at the end of each window. When a request r for file i departs the system, the histogram $F(x)$ of this monitor window is updated in the following steps:

- get the request's prefixed play duration D_r, and its encoding bitrate B_r;
- increment $F(x)$ by B_r for all $0 \le x \le access_i$, and by $B_r \cdot weight$ for all $access_i < x \le D_r - 1$;
- update $access_i$ considering D_r.

To adapt to the transient workload, we predict the next K requests by a geometrically discounted weighted sum of all the previously observed batches. The histogram $F(x)$ for the last K requests has the highest weight.

II. FlexSplitLard Strategy

Under FlexSplit, the first several servers in the cluster always have higher memory hit ratios than the last ones. The reason is that most of the accesses to short initial portion of the files are streamed by the first server from memory. Many of these sessions get terminated (as media workload analysis reveals), and fewer sub-requests go to the second server, and so on. As a result, the split duration gradually increases for latter servers. Less predictable and longer play durations streamed by the last servers decrease the memory hit ratios. One way to improve performance of the servers that stream media file "tails" is to exploit locality available in file accesses, and use different servers to stream different subset of the files, i.e. serve the file "tails" using the LARD strategy.

Therefore we propose a hybrid load balancing policy FlexSplitLard, which combines FlexSplit and LARD together. The splitting boundaries under FlexSplitLard are decided in the same way as FlexSplit, and the requests get service from the first n servers exactly same as in FlexSplit. For the requests falling into the servers after the nth server, they are not dispatched according to the play durations. Instead, they are served in the LARD manner among the left $(N-n)$ servers.

4 Performance Evaluation

I. Simulated workload

For workload generation, we use the publicly available, synthetic media workload generator *MediSyn* [11]. In our study we use synthetic workload that closely imitate parameters of real enterprise media server workloads [3].

The media file duration distribution can be summarized via following six classes: 10% of the files represent short videos 0-2min, 15% of the videos are 2-5min, 20% of the videos are 5-10min, 23% are 10-30min, 20% are 30-60min, and 12% of the videos are longer than 60 min. This distribution represent a media file duration mix that is typical for enterprise media workloads [3], where along with the short and medium videos (demos, news, and promotional materials) there is a representative set of long videos (training materials, lectures, and business events).

The file bit rates are defined by the following discrete distribution: 5% of the files are encoded at 56Kb/s, 20% - at 112Kb/s, 50% - at 256Kb/s, 25% at 500Kb/s.

Request arrivals are modeled by a Poisson process: a new request arrives each second on average.

The file popularity in our synthetic workload is defined by a generalized Zipf distribution [11] with $\alpha = 1.5$ and $k = 7$ in k-transformation. In summary, it has a fileset with 4000 files (with overall storage requirements of 207 GB), where 90% of the requests target 8% of the files. Correspondingly, these 8% of the most popular files have an overall combined size of 16.7 GB.

Number of Files	4000
Zipf α	1.5
Storage Requirement	207 GB
Number of Requests	1,089,245
Trace Duration	1 month

Table 1: Workload parameters used in simulation study.

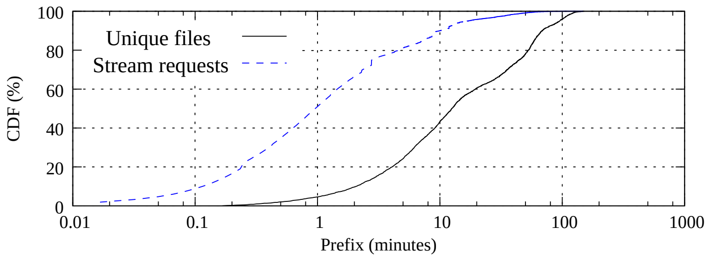

Figure 3: CDF of the media session (client aceess) durations and the original media file durations.

The evidence from media workload analysis [1, 2, 3] indicates that client demands are highly variable: some days (hours) exhibit 10-30 times higher load comparing to the typical daily (hourly) averages. In MediSyn, a user can specify a global diurnal pattern, which contains a set of bins. Each bin specifies a time period and the ratio of accesses in this bin. Our workload is defined by the 1-hour-long bins with load variation of up to 16 times between the "quiet" and "peak" load hours.

The requests to the streaming media server usually have very short play durations as illustrated in Figure 3. Although only 10% of unique files have play duration less than 2 minutes, 67% of the requests stay in the cluster for less than 2 minutes, and 90% of the requests stay in the cluster for less than 10 minutes.

We use 5-node media server cluster in our simulation study. The server capacity scaling rules for different encoding bit rates are similar to those measured using the experimental testbed described in Section 2. We use $cost_{X_i}^{disk}/cost_{X_i}^{memory} = 10$, i.e. the cost of disk access for files encoded at bit rate X_i is 10 times higher than the cost of the corresponding memory access. Finally, let the memory size of interest be 0.5 GB for each server.

II. Experimental results

In our performance study, we evaluate the following four load balancing solutions: *i) Round-Robin* strategy; *ii) LARD* strategy; *iii) FlexSplit* strategy; *iv)FlexSplitLard* strategy.

Commercial media servers are typically characterized by the number of concurrent streams a server can support without loosing a stream quality, i.e. until the real-time constraint of each stream can be met. Thus, a streaming media server must ensure that sufficient resources are available to serve a request (ideally, for the duration of the entire request). If not, the request should be rejected rather than allow it to consume resources that deliver little value to the end user, while simultaneously degrading the QoS to all clients already receiving content. To this end, a new request will be dropped if accepting it will overflow the system immediately or in the future. Dropping ratio is an important metric in evaluating the load balancing strategies. Table 2 provides the numbers of the rejected streams and the rejected bytes for each server and the overall cluster.

	Policy	Total	Node 1	Node 2	Node 3	Node 4	Node 5
Streams	RR	22458	4280	4254	4633	4604	4687
	LARD	2760	243	1108	187	817	405
	FlexSplit	1647	69	374	100	1025	79
	FlexSplitLard	1189	69	374	227	252	267
Bytes (GB)	RR	200	38	37	42	40	43
	LARD	22.4	2	10	1.6	5.8	2.8
	FlexSplit	48.8	0.9	6.8	2.4	35.6	3.1
	FlexSplitLard	28.3	0.9	6.8	5.8	5.8	9.0

Table 2: Number of rejected streams/bytes in the 5-node media cluster

As expected, RR has the worst performance. In overall it drops around 2.0% of the entire streams and 2.5% of the entire bytes given that the total number of streaming requests is 1,089,245 and the total number of bytes is around 7.9TB. However it performs equally in each server. The rejected numbers of both streams and bytes under LARD are only 1/10 of those under RR. FlexSplit drops 1/3 less number of streams compared with LARD, but as twice as much number of bytes. While it drops very little in the first server. Under FlexSplit the requests in the latter servers usually have longer play durations so that have larger request sizes. For example, the average number of bytes for each dropped stream in server 1 is 13MB, but it is 40MB in server 5. As a result, large number of request bytes might be dropped even if only small number of streams are dropped in the last several servers. In this way, FlexSplit favors small jobs to some extend, which is reasonable under high load to guarantee QoS. FlexSplitLard has the best performance considering dropping ratio metric. It has the exact same performance in the first 2 servers as FlexSplit, but much better performance in the last 3 servers. It only drops half number of streams of LARD, and similar number of bytes.

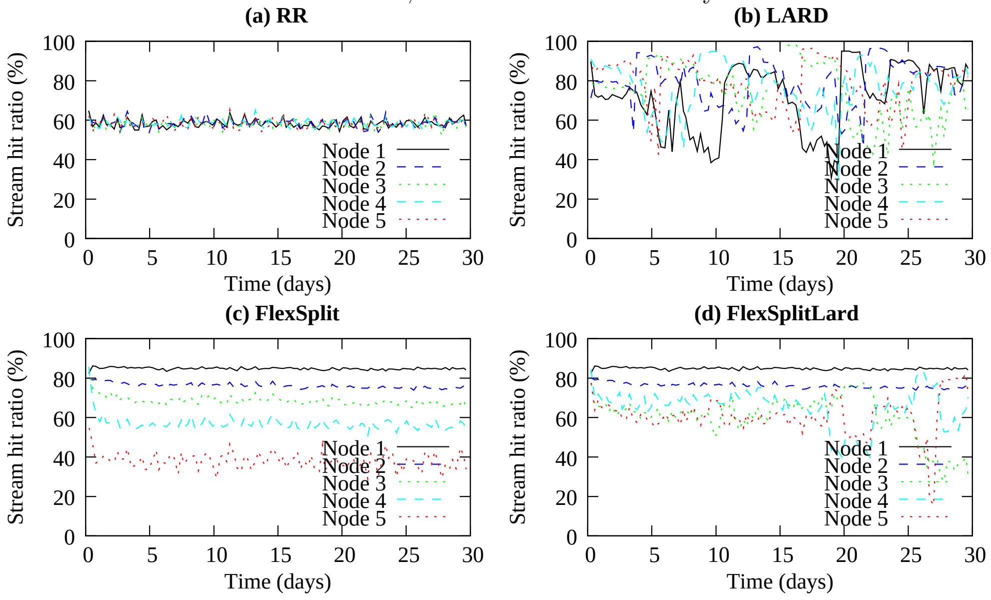

Figure 4: Average stream hit ratio for every 10K requests across the time.

Figure 4 evaluates the performance of each policy considering the memory hit ratios of the streams within each monitor window. RR has stable stream hit ratios as 60% across time as shown in Figure 4(a). While in Figure 4(b) the hit ratios under LARD is not stable at all. It can reaches up to 95%, and decreases down to 40% in one server. This is the influence under transient workload and the temporal locality. In overall, LARD still can achieve high hit ratios around 80% for all the servers. The hit ratios in FlexSplit are also stable but continuously reduce for latter servers. The first server has the highest hit ratio as about 85%, which is even better than LARD. Such numbers are about 76%, 68%, 52% and 40% for server 2, 3, 4 and 5 respectively. FlexSplitLard has much better overall average memory hit ratios in the last three servers compared with FlexSplit, which are between 62% to 68%.

The stable performance under FlexSplit is a great advantage over LARD since it provides predictable performance. Such predictability will benefit the system design in both capacity planning and QoS provisioning.

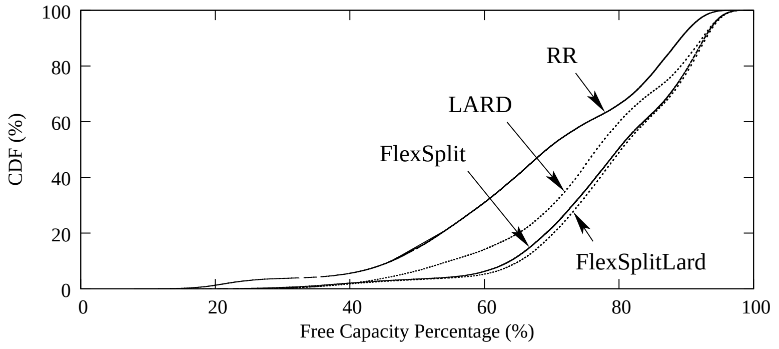

Figure 5: CDF of time percentage of system available capacity.

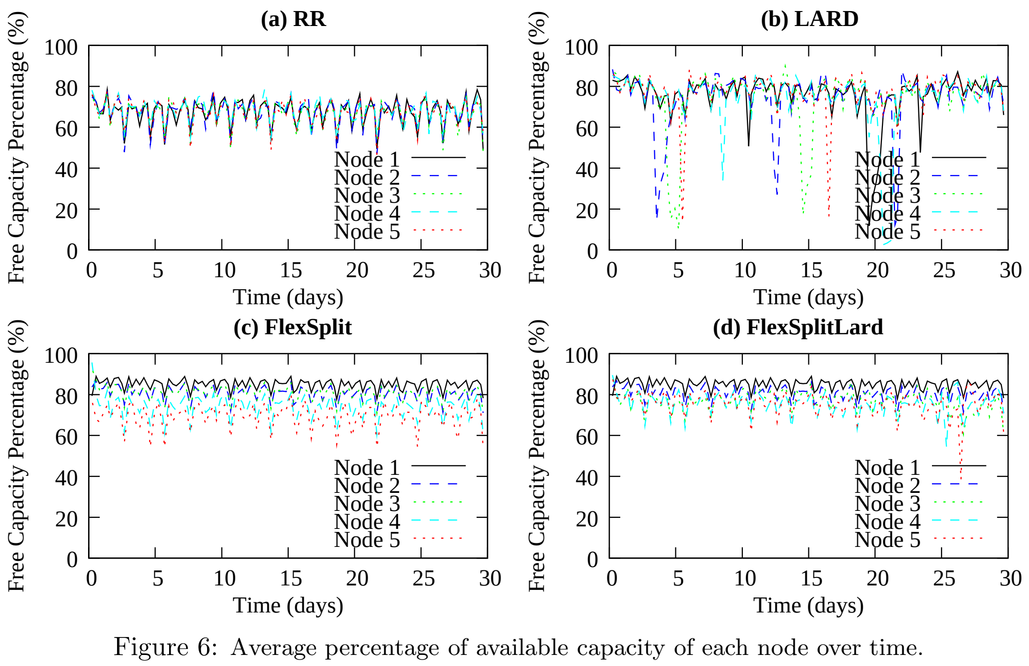

Figure 6: Average percentage of available capacity of each node over time.

Media workload measurements of existing media services indicate a “peak-to-mean” workload variance of more than one order of magnitude. Our workload is defined by the 1-hour-long bins with load variation of up to 16 times between the “quiet” and “peak” load hours. It motivates us to consider an additional metric of available capacity in the media cluster over time. While number of rejected streams reflects media cluster performance under the peak load, the metric of available capacity over time aims to evaluate overall media cluster performance over the entire simulation run.

The available capacity in the media cluster under different load balancing strategies is shown in Figure 5 and 6. Figure 5 gives the CDF of free capacity in the entire cluster. The x-axle is the ratio of the free capacity over the entire capacity in the cluster, and the y-axle is the cdf of time. A point (x, y) in the figure means that for y% of the experiment time, the cluster has less than x% available capacity. Therefore the higher line has less available capacity. As in the figure, RR has least available capacity, LARD is better than RR, FlexSplit is better than LARD, the FlexSplitLard performs best. But the improvement of FlexSplitLard compared with FlexSplit is small. More available capacity under FlexSplit and FlexSplitLard is another important benefit compared to LARD. This means that they have higher ability to server more requests, so that they can adapt to more bursty workloads better than LARD.

Figure 6 further illustrates the available capacity over time. This figure is similar as Figure 4 which indicates that the available capacity has a strong relation to the memory hit ratios. Again LARD has much burstier performance than the other policies, given that the difference of two conjunct monitor windows for available capacity percentage never exceeds 30% under RR and FlexSplit, but this value can reach 70% under LARD in day 5 and day 20.

5 Future Work

In this paper, we propose two novel load balancing strategies, FlexSplit and FlexSplitLard, which monitor and adaptively balance the incoming workload in a media cluster, aiming at efficiently utilize the combined cluster memory. By comparing with another two popular strategies RR and LARD via a simulation model, we illustrate that our policies can achieve lower number of dropped streams as well as higher available capacity. In the future work, we will further improve the adaptability of our policy by providing a more accurate cost function of the workload in term of the memory hit ratios in each server, so that the scaler *weight* can be decided on-line and no a priori knowledge is required.

References

[1] S. Acharya, B. Smith, P.Parnes. Characterizing User Access to Videos on the World Wide Web. In Proc. of ACM/SPIE Multimedia Computing and Networking. San Jose, CA, January 2000.

[2] Jussara Almeida, Jeffrey Krueger, Derek Eager, and Mary Vernon. Analysis of Educational Media Server Workloads. In *Proceedings of NOSSDAV*, June 2001.

[3] L. Cherkasova, M. Gupta. Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads. In Proc. ACM NOSSDAV 2002, May 2002.

[4] L. Cherkasova, L. Staley. Measuring the Capacity of a Streaming Media Server in a Utility Data Center Environment. Proc. of 10th ACM Multimedia, 2002.

[5] L. Cherkasova, W. Tang, A. Vahdat. MediaGuard: a model-based framework for building streaming media services. Proc. of Multimedia Computing and Networking 2005 (MMCN'05).

[6] Z. Ge, P. Ji, P. Shenoy. A Demand Adaptive and Locality Aware (DALA) Streaming Media Cluster Architecture. Proc. of ACM NOSSDAV, 2002.

[7] Yang Guo, Zihui Ge, Bhuvan Urgaonkar, Prashant Shenoy and Don Towsley, "Bandwidth and Space-Constrained Caching Strategies for Cluster-Based Streaming Proxies", Proceeding of WCW 2003, September 2003, Hawthorne, New York, USA.

[8] D. Luperello, S. Mukherjee, and S. Paul. Streaming Media Traffic: an Empirical Study. Proc. of Web Caching Workshop (WCW'02), June 2002.

[9] V.Pai, M.Aron, G.Banga, M.Svendsen, P.Drushel, W. Zwaenepoel, E.Nahum: Locality-Aware Request Distribution in Cluster-Based Network Servers. Proc. of the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII), ACM SIGPLAN, 1998.

[10] Subhabrata Sen, Jennifer Rexford, and Don Towsley. Proxy Prefix Caching for Multimedia Streams. In *Proceedings of INFOCOM*, March 1999.

[11] W. Tang, Y. Fu, L. Cherkasova, A. Vahdat. MediSyn: A Synthetic Streaming Media Service Workload Generator. Proc. of ACM NOSSDAV, 2003.

[12] Q. Zhang, A. Riska, W. Sun, E. Smirni, G. Ciardo. Workload-Aware Load Balancing for Clustered Web Servers.In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pp. 219-233, Vol. 16(3), Mar. 2005.