Providing Resource Allocation and Performance Isolation in a Shared Streaming-Media Hosting Service

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

Wenting Tang
Hewlett-Packard Laboratories
1501 Page Mill Road,
Palo Alto, CA 94303, USA
wenting.tang@hp.com

Abstract *The trend toward media content hosting is seeing a significant growth as more rich media is used in the enterprise environment and as it becomes mission critical for businesses. A shared media hosting service supports the illusion that each hosted service has its own media server, when, in reality, multiple "logical hosts" may share one physical host. For such a shared media hosting service, the ability to guarantee a specified share of server resources to a particular hosted service is very important. We present SharedMediaGuard - a shared media hosting infrastructure that can efficiently allocate the predefined shares of server resources to the hosted media services. The proposed solution is based on a unified cost function that uses a single value to reflect the combined resource requirement such as CPU, bandwidth and memory necessary to support a particular media stream depending on the stream bit rate and type of access (memory file access or disk file access). Our evaluation of SharedMediaGuard compared to the traditional, disk-based allocation strategy that assumes all content must be served from disk reveals a factor of two improvement in server throughput while providing performance isolation and QoS guarantees among the hosted services.*

Categories and Subject Descriptors*: H.5.1 Multimedia Information Systems.*
General Terms*: Measurement, Performance, Design.*
Keywords*: Shared media hosting, media server capacity, benchmarking, measurement, SLAs, admission control, performance isolation, QoS guarantees, simulation.*

1. INTRODUCTION

Media content hosting is an increasingly common practice. In media content hosting, providers who have a large amount of resources offer to store and provide access to media files from institutions, companies and individuals who are looking for a cost efficient, "no hassle" solution. A shared media hosting service supports the illusion that each hosted servicet has its own media server, when, in reality, multiple "logical hosts" may share one physical host. When each of the hosted media services is small, i.e. it publishes only a small number of media files and the traffic to such a small media site is somewhat light, the service provider might offer the "best effort service" (using a flat subscription fee) without a performance isolation among the sites or QoS guarantees to the hosted sites. Typically, this simple arrangement leads to a well utilized infrastructure while still providing good performance results.

However, the situation changes when, for example, two media sites are sharing the same media server, and each is paying for 50% of the server resources. For such a shared media service, the ability to enforce the performance isolation among the services and to provide the fair share of resource usage is very important. A service provider is required to comply with *Service Level Agreement (SLA)* – a contract where a service provider is promising that X_s% of system resources will be allocated to a designated media service s. In our paper, we assume that files are encoded at constant bit rates (that is typical for commercial media sites). Thus, for a given workload, it is easy to determine whether sufficient network bandwidth is available. However, it is more difficult to determine what amount of CPU, memory, and disk resources are needed to process a given workload with specified performance requirements.

The problem of allocating X_s% of system capacity to a designated media service s is inherently similar to an admission control problem: we need to be able to admit a new request to service s when the utilized server capacity by service s is below a threshold X_s% and to reject the request otherwise. Commercial media server solutions do not have a "built-in" admission control that can prevent server overload or allocate a predefined fraction of server resources to a particular service. Moreover, streaming media server does not have a single system resource which can be observed for evaluating the currently available/used server capacity: the CPU and disk utilization as well as the bandwidth usage are highly dependent on the workload type. Current media servers continue to admit new client requests and provide degraded quality of service by sharing the available server resources among the admitted streams.

In recent work [3], a set of benchmarks for measuring the basic capacities of streaming media systems was proposed. The benchmarks allow one to derive the scaling rules of server capacity for delivering media files which are: *i)* encoded at different bit rates, *ii)* streamed from memory vs disk. From the set of basic benchmark measurements, we can derive a *cost function* which uses a single value to reflect the combined resource requirement such as CPU, bandwidth and memory (i.e. a corresponding *fraction* of system resources) necessary to support a particular media stream depending on the stream bit rate and type of access (memory file access or disk file access).

In this paper, we design a shared media hosting infrastructure, called *SharedMediaGuard*, which is based on a unified *cost*

SAC'04, March 14–17, 2004, Nicosia, Cyprus.

function and that can efficiently allocate the predefined shares of server resources to the hosted media services. For streaming media requests, an additional complexity consists in determining the level of available system resources as a function of time. For long-lasting real-time requests, it is insufficient to simply consider the current level of, for instance, available memory or idle CPU. Rather, sometimes complex interactions with memory caching, disk and network bandwidth, and workload specific features, etc. must be considered in evaluating whether a particular request can be satisfied with particular QoS characteristics for the entire duration of the request. In order to estimate the cost of a new request, we evaluate whether a new request will be streaming data from memory or disk, and whether the acceptance of this new request may violate the overall QoS guarantees of already accepted requests at any point in the future.

We evaluate the efficiency of the new strategy using a simulation model and a set of synthetic workloads closely imitating parameters of real enterprise media server workloads: *SharedMediaGuard* provides almost two times better throughput both in the number of accepted requests and in delivered bandwidth against the traditional, *disk-based* resource allocation strategy. By using the *SharedMediaGuard* framework, where each media request is characterized via a cost function defining a fraction of system resources needed to support the corresponding media stream, we are able to provide a powerful mechanism for tailored allocation of shared resources with the performance isolation among the services, and the support of the required SLAs.

The remainder of the paper presents our results in more detail.

2. RELATED WORK

Commercial media systems may contain hundreds to thousands of clients. Given the real-time requirements of each client, a multimedia server has to employ admission control algorithms to decide whether a new client request can be admitted without violating the quality of service requirements of the already accepted requests. There has been a large body of research work on admission control algorithms. Existing admission control schemes were mostly designed for disk subsystems and can be classified by the level of QoS provided to the clients [15]. Most of the papers are devoted to storage and retrieval of variable bit rate data. *Deterministic* admission control schemes provide strict QoS guarantees, i.e. the continuous playback requirements should never be violated for the entire service duration. The corresponding algorithms are characterized by the worst case assumptions regarding the service time from disk [6, 12, 14, 16, 17, 4]. In [15, 8, 1, 7], *statistical* ac-algorithms are designed which provide probabilistic QoS guarantees instead of deterministic ones, resulting in higher resource utilization due to statistical multiplexing gain.

Continuous media file servers require that several system resources be reserved in order to guarantee timely delivery of the data to clients. These resources include disk, network and processor bandwidth. A key component of determining the amount of a resource to reserve is characterizing each streams' bandwidth. In [9, 11, 5] the admission control is examined from the point of *network bandwidth allocation* on the server side.

Our work proceeds further and addresses a critical system resource that has not received much attention in the past: the impact of main memory on media system performance. Since many popular media files have footprints on the order of 10 MB and smaller, and at the same time, modern servers might have up to 4 GB of main memory, it means that most of the accesses to popular media objects can be served from memory, even when a media server relies on a traditional file system support and does not have additional application level caching. Our shared media hosting framework exploits the benefits of system level caching in delivering media applications for typical streaming media workloads.

3. BENCHMARKING A MEDIA SERVER CAPACITY

Commercial media servers are 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. In paper [3], two basic benchmarks were introduced 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.

Each of these benchmarks consists of a set of sub-benchmarks with media content encoded at a different bit rate (in our performance study, we used six bit rates representing the typical Internet audience: 28 Kb/s, 56 Kb/s, 112 Kb/s, 256 Kb/s, 350 Kb/s, and 500 Kb/s. Clearly, the set of benchmarked encoding bit rates can be customized according to targeted workload profile). Using an experimental testbed and a proposed set of basic benchmarks, we measured capacity and scaling rules of a media server running RealServer 8.0 from RealNetworks. The configuration and the system parameters of our experimental setup are specially chosen to avoid some trivial bottlenecks when delivering multimedia applications such as limiting I/O bandwidth between the server and the storage system, or limiting network bandwidth between the server and the clients.

The measurement results show that the scaling rules for server capacity when multiple media streams are encoded at different bit rates are non-linear. For example, the difference between the highest and lowest bit rate of media streams used in our experiments is 18 times. However, the difference in maximum number of concurrent streams a server is capable of supporting for corresponding bit rates is only around 9 times for a *Single File Benchmark*, and 10 times for a *Unique Files Benchmark*. The media server performance is 3 times higher (for some disk/file subsystem up to 7 times higher) under the *Single File Benchmark* than under the *Unique Files Benchmark*. This quantifies the performance benefits for multimedia applications when media streams are delivered from memory.

Using a set of basic benchmark measurements, we 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):

- $cost^{disk}_{X_i}$ - a value of cost function for a stream with disk access to a file encoded at X_i Kb/s. If we define the media server capacity being equal to 1, the cost function is computed as $cost^{disk}_{X_i} = 1/N^{Unique}_{X_i}$, where $N^{unique}_{X_i}$ - the maximum measured server capacity in concurrent streams under the *Unique File Benchmark* for X_i Kb/sencoding,
- $cost^{memory}_{X_i}$ - a value of cost function for a stream with memory access to a file encoded at X_i Kb/s. Let $N^{single}_{X_i}$

- the maximum measured server capacity in concurrent streams under the *Single File Benchmark* for a file encoded at X_i Kb/s. Then the cost function is computed as $cost^{memory}_{X_i} = (N^{unique}_{X_i} - 1)/(N^{unique}_{X_i} \times (N^{single}_{X_i} - 1))$.

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 ($X_w \subseteq X$),
- $N^{memory}_{X_{w_i}}$- 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

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

If $Demand \leq 1$ then the media server operates within its capacity, and the difference $1 - Demand$ defines the amount of available server capacity.

4. *SHAREDMEDIAGUARD* INFRASTRUCTURE

Let $s_1, ..., s_k$ be the media services supported by the streaming media server. Let the SLA requirements for the hosted services are defined in the following way: media service s_i is guaranteed an allocation of X_i% of media server capacity and $X_1 + ... + X_k \leq 100\%$. The main goal of the *SharedMediaGuard* mechanism is to keep the allocated fraction of a media server resources to a corresponding service s_i below or equal to its specified threshold X_s%. *SharedMediaGuard* performs two main procedures when evaluating whether a new request $r^f_{s_i\_new}$ to a media service s_i can be accepted:

- *Resource Availability Check*: during this procedure, a cost of a new request $r^f_{s_i\_new}$ is evaluated. To achieve this goal we evaluate the memory (file buffer cache) state to identify whether a prefix of requested file f is residing in memory, and whether request $r^f_{s_i\_new}$ will have a cost of accessing memory or disk correspondingly. Then, *SharedMediaGuard* checks whether in the current time, the media service s_i has enough available capacity to accommodate the resource requirements of new request $r^f_{s_i\_new}$.
- *QoS Guarantees Check:* When there is enough currently available server capacity to admit a new request $r^f_{s_i\_new}$ for a particular service s_i , the *SharedMediaGuard* mechanism still needs to ensure that the acceptance of request $r^f_{s_i\_new}$ will not violate the QoS guarantees of already accepted requests in the system over their lifetime and that the media server will not enter an overloaded state at any point in the future.

4.1 Resource Availability Check

In order to assign the *cost* to a new media request, we need to evaluate whether this 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 a sequence of accesses to the same file is issued closely to each other on a time scale, then the first access may read a file from disk, while the subsequent requests may be accessing the corresponding file prefix from memory.

The basic idea of computing the request access type is as follows. Let $Size^{mem}$ be the size of memory in bytes [1]. For each request r in the media server access log, we have the 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 the time when a stream initiated by request r is terminated.

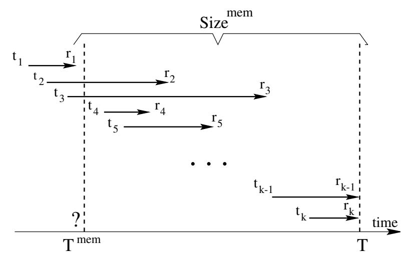

Figure 1: Memory state computation example.

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. In such a way, we can identify whether request r will stream file f (or some portion of it) from memory.

To realize this idea efficiently, we designed an induction-based algorithm for computing the memory state at any given time. Let T_{cur} be a current time corresponding to a new request $r^f_{s_i\_new}$ arrival. Let T_{prev} denote the time of the previous arrival event, and let T^{mem}_{prev} be the previously computed time such that the sum of bytes stored in memory between T^{mem}_{prev} and T_{prev} is equal to $Size^{mem}$ as shown in Figure 2.

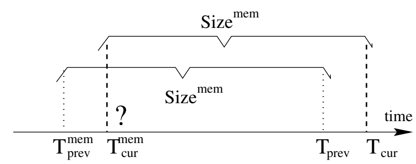

Figure 2: Induction-based memory state computation.

At time point T_{cur}, we compute

- an updated time T^{mem}_{cur} such that the sum of bytes stored in memory between T^{mem}_{cur} and T_{cur} is equal to $Size^{mem}$;
- an updated information on file segments stored in memory in order to determine the cost of new request $r^f_{s_i\_new}$.

4.2 QoS Guarantees Check

For long-lasting streaming media requests, an additional complexity consists in determining the level of available system resources as a function of time. When service s_i has enough *currently* available server capacity to admit a new request $r^f_{s_i\_new}$, we still need to ensure that the acceptance of request $r^f_{s_i\_new}$ will not violate the overall QoS guarantees of already accepted requests over their lifetime.

SharedMediaGuard assesses the following possible situations which may result in increased server resource usage.

- Let a new request $r^f_{s_i\_new}$ be a disk access. In this case, there is a continuous stream of new bytes transferred from

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

disk to memory (number of new bytes is defined by the file encoding bit rate). It may result in replacement of some "old" file segments in memory, for example, let it be some segments of file $\hat{f}$. If there is an active request $r^{\hat{f}}$ which reads the corresponding file segments from memory and has the cost of memory access then once the corresponding segments of file $\hat{f}$ are evicted (replaced) from memory, the request $r^{\hat{f}}$ will read the corresponding segments of file $\hat{f}$ from disk with increased cost of disk access (we will call that request $r^{\hat{f}}$ is *downgraded*), i.e. the acceptance of new request $r^f_{s_i\_new}$ will lead to an increased cost of request $r^{\hat{f}}$ in the future.

- Let a new request $r^f_{s_i\_new}$ be a memory access. Then we need to assess whether request $r^f_{s_i\_new}$ has the "memory" cost during its life or whether the corresponding segments of file f may be evicted by already accepted active disk requests some time in the future, and request $r^f_{s_i\_new}$ will read the corresponding segments of file f from disk with increased cost of disk access.

We need to assess such situations whenever they may occur in the future for accepted "memory" requests and evaluate whether the increased cost of downgraded requests can be offset by the *overall available capacity* of the server in the corresponding time points.

REMARK: We can enforce a *stronger* performance isolation among the services by evaluating whether the increased cost of downgraded requests can be offset by the *available capacity* of their *corresponding services*. However, it may lead to a "starvation-like" problem among the services under certain scenarios. For example, let service s_1 have a 50% share in server capacity, and it is currently fully utilized by already accepted requests. Let these requests of service s_1 stream data from memory, and let they occupy all the space in file buffer cache of the media server. Let service s_2 have another 50% of overall server capacity, and its share is fully available. Let us consider whether service s_2 can accept a new request $r^f_{s_2\_new}$ according to the "strong" performance isolation definition. First of all, request $r^f_{s_2\_new}$ will stream its data from disk (since the server file buffer cache is occupied by files from s_1). Second, if $r^f_{s_2\_new}$ is accepted then it will inevitably downgrade some of the requests from s_1 over time. However, service s_1 does not have available capacity (in its share) to offset the increased cost of its downgraded requests. Hence, service s_2 may not accept a new request according to the "strong" performance isolation definition, even when the overall server capacity is plentiful to offset the increased cost of downgraded requests from s_1. To avoid this situation, we chosed to control the performance isolation between the services only at the *resource availability check* of the algorithm, while using the overall available server capacity to offset the increased cost of downgraded requests at the *QoS guarantees check* of the algorithm. *END of REMARK*

The main idea of our algorithm on QoS validation is as follows. We partition all the active requests in two groups: *i)* active memory requests, i.e. the requests which have a memory access cost; and *ii)* active disk requests, i.e. the requests which have a disk access cost.

Active memory requests access their file segments in memory. Thus, they do not bring new bytes to memory, they only refresh the accessed file segments' timestamps with the current time. Only active disk requests bring "new" bytes from disk to memory and evict the corresponding amount of "oldest" bytes from memory. *SharedMediaGuard* identifies the bytes in memory with the oldest timestamp (let it be $T^{act\_m}_{cur}$) which are read by some of the active memory requests. Thus, all the bytes stored in memory prior to $T^{act\_m}_{cur}$ can be safely replaced in memory (as depicted in Figure 3) without impacting any active memory requests.

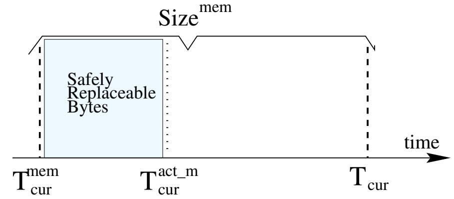

Figure 3: Safely Replaceable Bytes in Memory.

Using the information about file encoding bit rates as well as the future termination times for active disk requests we compute a time duration during which the active disk requests will either transfer from disk to memory the amount of bytes equal to *SafelyReplBytes*(T_{cur}) or all of them will terminate. By repeating this process in the corresponding future points, we identify whether active disk requests are always evicting only "safely replaceable bytes" in memory or some of the active memory requests have to be downgraded. In latter case, *SharedMediaGuard* evaluates whether the increased cost of downgraded requests can be offset by the available server capacity at these time points.

5. PERFORMANCE EVALUATION

In order to evaluate the performance benefits of the new shared media hosting infrastructure, we built a simulation model of *SharedMediaGuard* in Java.

We compared *SharedMediaGuard* performance against the *disk-based* shared hosting service that pessimistically assumes that all accesses must go to disk. The *disk-based* shared hosting service measures the media server capacity as a number of concurrent streams the disk subsystem is capable of supporting. The disk-based shared hosting policy can be used to allocate the predefined share of service resources to a particular service and to provide the performance isolation among the services. When a new request $r^f_{s_i\_new}$ to a media service s_i arrives, it is assigned the cost of disk access. If service s_i has enough available capacity (in its share) to accommodate the resource requirements of new request $r^f_{s_i\_new}$, it is accepted. Thus, the disk-based policy does not take into account the impact of main memory on media system performance.

For workload generation, we use the publicly available, synthetic media workload generator *MediSyn* [13]. We perform a sensitivity study, using two services s_1 and s_2 sharing the same media server allocated in the following way between the services:

- 70% of the media server capacity is allocated to s_1;
- 30% of the media server capacity is allocated to s_2.

Workloads for s_1 and s_2 are imitating parameters of the real enterprise media server workloads [2]. The overall statistics for *Workload 1* of service s_1 and *Workload 2* of service s_2, used in the study, is summarized in Table 1:

	Workload 1	**Workload 2**
Number of Files	800	800
Zipf α	1.34	1.22
Storage Requirement	41 GB	41 GB
Number of Requests	41,703	24,159

Table 1: Workload parameters used in simulation study.

Both synthetic workloads have the same media file duration distribution, which can be briefly summarized via six following

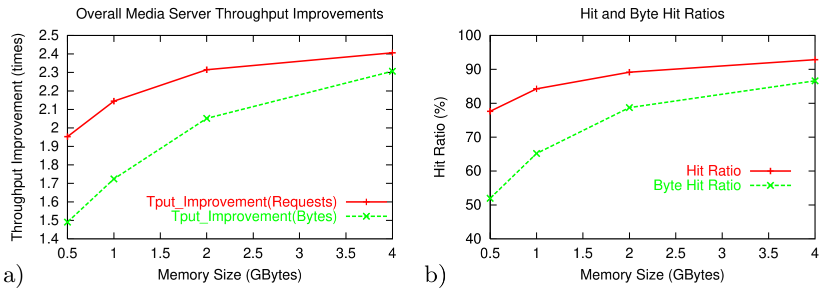

Figure 4: a) Normalized throughput improvements under *SharedMediaGuard* compared to the *disk-based* strategy, b) hit and byte hit ratio for accepted session served from memory under *SharedMediaGuard*.

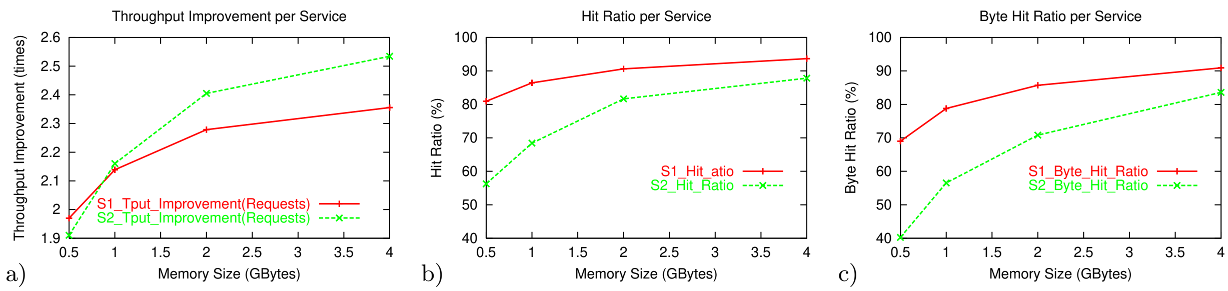

Figure 5: a) Normalized throughput improvements under *SharedMediaGuard* compared to the *disk-based* strategy (per service s_1 and s_2), b) hit ratio per service s_1 and s_2 under *SharedMediaGuard* c) byte hit ratio per service s_1 and s_2 under *SharedMediaGuard*.

classes: 20% of the files represent short videos 0-2min, 10% of videos are 2-5min, 13% of videos are 5-10min, 23% are 10-30min, 21% are 30-60min, and 13% of videos are longer than 60 min.

The file bitrates are defined by the following discrete distribution: 5% of the files are encoded at 56Kb/s, 20% - at 112Kb/s, 50% - at 256Kb/s, 20% - at 350Kb/s, and 5% - at 500Kb/s.

The file popularity is defined by a Zipf-like distribution with α shown in Table 1. In summary, *Workload 1* has a higher locality of references than *Workload 2:* 90% of requests target 10% of files in *Workload 1* compared to 90% of requests targeting 20% of files in *Workload 2*. Correspondingly, 10% of most popular files in *Workload 1* have overall combined size of $3.8GB$, while 20% of most popular files in *Workload 2* us $7.5GB$ of storage space.

Session arrivals are modeled by a Poisson process with rate corresponding to the consistent media server overload (on average, there is a request arrival in each second).

We performed a set of simulations for a media server with a different memory size of $0.5GB$, $1GB$, $2GB$, and $4GB$. While 4 GB might be an unrealistic parameter for file buffer cache size, we are interested to see the dependence of a performance gain due to increased memory size. We defined the server capacity and the cost functions similar to those measured using the experimental testbed described in Section 3. We use $cost^{disk}_{X_i}/cost^{memory}_{X_i} = 3$, i.e. the cost of disk access is 3 times higher than the cost of the corresponding memory access to the same file.

The overall performance results for simulated media server hosting services s_1 and s_2 are shown in Figures 4. Figure 4 a) represents the normalized media server throughput improvements under the *SharedMediaGuard* strategy compared to the *disk-based* allocation strategy using two metrics: the number of accepted requests and the total number of transferred bytes. *SharedMediaGuard* significantly outperforms the *disk-based* strategy. For instance, for a file buffer cache of 2 GB, *SharedMediaGuard* shows a factor of two improvement in overall media server throughput (where 70% of server capacity is allocated to service s_1 and 30% - to service s_2) for both metrics, while providing the desirable QoS guarantees and performance isolation for hosted media services. It reveals that the media server throughput can be significantly improved via main memory support even for media server with relatively small memory size. The memory increase does not result in a "linear" performance gain as shown in Figures 4 a). For example, the memory increase from 2 GB to 4 GB results in less than 10% of additional performance gain in the server throughput.

Figure 4 b) shows the overall hit and byte hit ratio for sessions accepted by the media server and served from memory. Even for a relatively small file buffer cache (such as 0.5 GB), more than 75% of the sessions are served from memory. These sessions are responsible for more than 50% of bytes transferred by the media server.

Figure 5 a) represents the normalized throughput improvements under *SharedMediaGuard* compared to the *disk-based* allocation strategy per service (s_1 and s_2) using the number of accepted requests. As already expected, *SharedMediaGuard* more than 2 times outperforms the *disk-based* strategy for both services. Interestingly, service s_2 has a higher performance gain increase for larger buffer cache than the service s_1. It can be explained by the fact that *Workload 2* has less locality and, therefore, a larger "footprint" for the most frequent files compared to *Workload 1.* As a result, service s_2 can better "utilize" a larger size memory. Figures 5 b), c) show the hit ratio and byte hit ratio for sessions accepted by each of the services s_1 and s_2 and served from memory. For a relatively small memory size, service s_1 has a significantly higher hit ratio than service s_2 (which can be explained by the higher reference locality in service s_1). As we can see, for a larger memory size, service s_1 has a very modest performance gain due to additional memory (most of the misses in s_1 are the "cold" misses, and can not be improved by the additional memory). In contrast, service s_2 demonstrates a much higher performance gain on larger size memory, clearly leveraging the available resource sharing to the

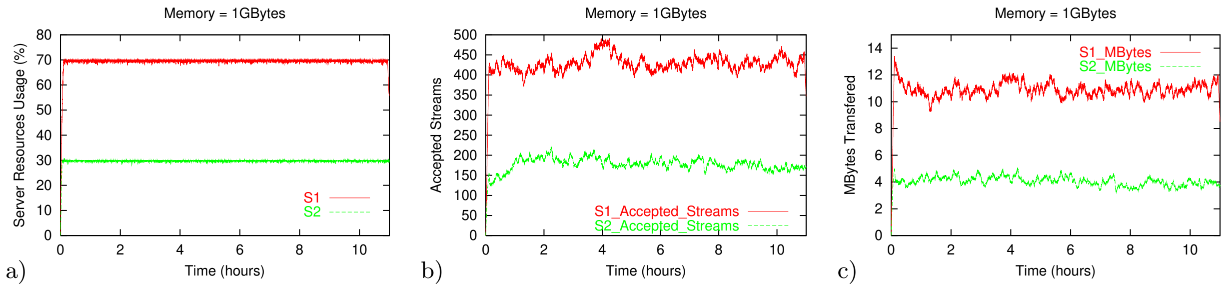

Figure 6: ***SharedMediaGuard*: a) Server capacity allocation between two services over time, b) the number of accepted sessions by each service over time, c) the number of MBytes transferred by each service over time.**

benefit of multiple services.

Figure 6 a) represents how the server capacity was allocated between two services over time: service s_1 is allocated 70% of server capacity and service s_2 had received 30% of server resources. Thus, indeed *SharedMediaGuard* enables a tailored allocation of shared resources with QoS guarantees and performance isolation for hosted media services.

Figure 6 b) demonstrates the number of accepted and served clients sessions for both services over time. While both services are utilizing a fixed fraction of the server capacity over time (70% - service s_1 and 30% - service s_2), we can see that the number of accepted and processed sessions is far from being fixed: it varies within 35% for each of considered services. These results are expected because the cost for serving the session from disk is 3 times higher than from memory. Media services with a high reference locality may benefit in significantly increased number of processed client requests due to efficient memory usage while utilizing the same fraction of media server capacity. Figure 6 c) demonstrates the number of transferred MBytes per service. Similarly, it is variable for each service, because of the varying number of accepted clients requests by each service as well as a broad variety of encoding bit rates for corresponding media content.

6. CONCLUSION AND FUTURE WORK

Delivering performance isolation to competing services while at the same time leveraging available resource sharing to the benefit of multiple services is an interesting and difficult problem that we address in the context of a shared media hosting service.

Different system resources are limiting a media server performance under different workload parameters. When a media server processes streams from memory, the server performance is CPU bounded. While when the sessions are delivered from disk, the server performance is disk-bound. Under the "mixed" workload, neither a CPU nor a disk are system bottlenecks. Thus one of the difficult problems for real-time applications like streaming media server is that monitoring of a single system resource can not be used for evaluating the currently available/used server capacity.

In this paper, we designed a shared media hosting infrastructure, called *SharedMediaGuard*, which is based on a unified *cost* function using a single value to reflect the combined resource requirement such as CPU, bandwidth and memory necessary to support a particular media stream depending on the stream bit rate and type of access (memory file access or disk file access). Our evaluation of *SharedMediaGuard* compared to the traditional, *disk-based* allocation strategy that assumes all content must be served from disk reveals a factor of two improvement in server throughput while providing performance isolation and QoS guarantees among the hosted services.

In the future, we plan to extend the *SharedMediaGuard* framework to a cluster solution. Closely related problem is an assignment of hosted services to the cluster nodes in such a way that the overall system throughput under *SharedMediaGuard* is maximized.

7. REFERENCES

[1] E. Biersack, F. Thiesse. Statistical Admission Control in Video Servers with Constant Data Length Retrieval of VBR Streams. Proc. of the 3d Intl. Conference on Multimedia Modeling, France, 1996.

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

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

[4] J. Dengler, C. Bernhardt, E. Biersack. Deterministic Admission Control Strategies in Video Servers with Variable Bit Rate. Proc. of European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), Germany, 1996.

[5] Y. Fu, A. Vahdat. SLA-Based Distributed Resource Allocation for Streaming Hosting Systems. Proc. 7th International Workshop on Web Content Caching and Distribution, 2002.

[6] J. Gemmell, S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. ACM Transactions on Information Systems, 10(1), 1992.

[7] X. Jiang, P. Mohapatra. Efficient admission control algorithms for multimedia servers. J. ACM Multimedia Systems, vol. 7(4), July, 1999.

[8] E. Knightly, H. Zhang. Traffic Characterization and Switch Utilization Using Deterministic Bounding Interval Dependent Traffic Models. Proc. of IEEE INFOCOM'95, Boston 1995.

[9] E. Knightly, D. Wrege, J. Lieberherr, H. Zhang. Fundamental Limits and Tradeoffs of Providing Deterministic Guarantees to VBR Video Traffic. Proc. of ACM SIGMETRICS'95.

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

[11] D. Makaroff, G.Neufeld, N. Hutchinson. Network Bandwidth Allocation and Admission Control for a Continuous Media File Server. Proc. of the 6th Intl. Workshop on Interactive Distributed Multimedia Systems and Telecommunications Services, Toulouse, France, 1999.

[12] N. Reddy, J. Willie. Disk Scheduling in Multimedia I/O System. Proc. of ACM Multimedia, Anaheim, 1993.

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

[14] F. Tobagi, J. Pang, R. Baird, M.Gang. Streaming RAID: A Disk Storage for Video and Audio Files. Proc. of ACM Multimedia, Anaheim, 1993.

[15] H. Vin, P. Goyal, A. Goyal, A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In Proc. of ACM Multimedia, San Francisco, 1994.

[16] H. Vin, P. Rangan. Designing a Multi-User HDTV Storage Server. IEEE J. on Selected Areas in Communications, 11(1), Jan., 1993.

[17] P. Yu, M. Chen, D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Mutimedia Storage Management. Proc. of ACM NOSSDAV, San Diego, 1992.