Exploiting Virtualization for Delivering Cloud-based IPTV Services

Vaneet Aggarwal, Xu Chen, Vijay Gopalakrishnan, Rittwik Jana, K. K. Ramakrishnan, Vinay A. Vaishampayan
AT&T Labs – Research, 180 Park Ave, Florham Park, NJ, 07932

***Abstract*—Cloud computing is a new infrastructure environment that delivers on the promise of supporting on-demand services in a flexible manner by scheduling bandwidth, storage and compute resources on the fly. IPTV services like Video On Demand (VoD) and Live broadcast TV requires substantial bandwidth and compute resources to meet the real time requirements and to handle the very bursty resource requirements for each of these services. To meet the needs of the bursts of requests, each with a deadline constraint for both VoD and LiveTV channel changes, we propose a resource provisioning framework that allows these services to co-exist on a common infrastructure by taking advantage of virtualization. We propose an optimal algorithm that provides the minimum number of servers needed to fulfill all requests for these services. We prove this optimality in a general setting for any number of services with general deadline constraints. By using real world data from an operational IPTV environment, our results show that anticipating and thereby enabling the delaying of VoD requests by up to 30 seconds gives significant resource savings even under conservative environmental assumptions. We also experiment with different scenarios (by varying the deadline constraints, changing the peak to average ratios of the constituent services) to compute the overall savings.**

I. Introduction

As IPTV-based television becomes more popular, one of the biggest challenges service providers face today is to support users' voracious appetite for entertainment video across the various IPTV services (Live TV, Video on Demand etc). To cater to the different users and their needs, providers provision for the peak demands of each service. However, since the resource demands of none of these services is constantly or even concurrently at the peak, provisioning for the peak is sub-optimal.

In this paper, we investigate the potential of utilizing virtualization to support multiple services like Video On Demand (VoD) and Live broadcast TV (LiveTV). We explore how we can carefully configure the cloud infrastructure in real time to sustain the large scale bandwidth and computation intensive IPTV applications (e.g. LiveTV instant channel changes (ICC) and VoD requests). In IPTV, there is both a steady state and transient traffic demand [1]. Transient bandwidth demand for LiveTV comes from clients switching channels. This transient and highly bursty traffic demand can be significant in terms of both bandwidth and server I/O capacity. The challenge is that we currently have huge server farms for serving individual applications that have to be scaled as the number of users increases. In this paper, we focus on dedicated servers for LiveTV ICC and VoD. Our intent is to study how to efficiently minimize the number of servers required by using virtualization within a cloud infrastructure to replace dedicated application servers.

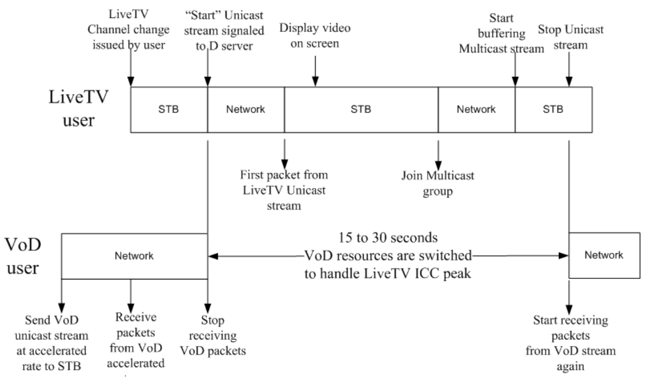
LiveTV Channel change issued by user
"Start" Unicast stream signaled to D server
Display video on screen
Start buffering Multicast stream
Stop Unicast stream
LiveTV user
STB
Network
STB
Network
STB
First packet from LiveTV Unicast stream
Join Multicast group
VoD user
Network
15 to 30 seconds VoD resources are switched to handle LiveTV ICC peak
Network
Send VoD unicast stream at accelerated rate to STB
Receive packets from VoD accelerated
Stop receiving VoD packets
Start receiving packets from VoD stream again

Fig. 1. LiveTV ICC and VoD packet buffering timeline

When there are multiple services (in this context, VoD and LiveTV ICC) that coexist, and if some services have very high peak to average ratios, multiplexing can help to reduce the total resource requirements. LiveTV ICC is emblematic of such a service, with a large amount of correlated requests arriving periodically. In this paper, we adapt the servicing of VoD requests to the predictably bursty LiveTV ICC requests using the cloud platforms capability to provision resources dynamically. We obtain the minimum number of servers (i.e., virtual machines) required by carefully studying the tradeoffs offered for a composite service environment (with a goal of meeting the needs of the peak of the sums) as opposed to the sum of the peaks (when the services are provisioned independently). There is a large peak to average ratio for LiveTV ICC. The real time aspect of entertainment content requires us to provision resources to handle peak demand. Moreover, the ICC demand is very peaky, using resources only for a very short period of time. VoD on the other hand has a relatively steady load and imposes delay bounds.

In our virtualized environment, ICC is managed by a set of VMs. The number of such VMs created would be driven by the predictor described above (note that a (small) number of VMs would typically be assigned to each distinct channel). Similarly, for the VoD service, we would configure a number of VMs based on the currently active VoD sessions, and would be adapted to meet user demand. When a physical server complex is shared for these services, it is desirable that we minimize the total number of VMs deployed (thereby the resources used) to satisfy all these requests. The provisioning

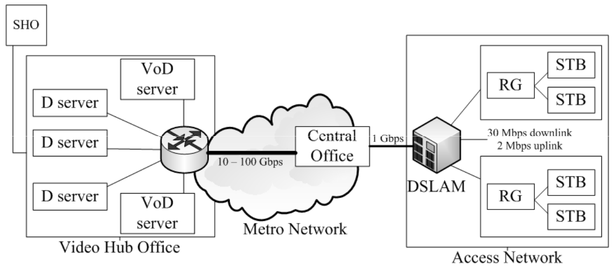

Fig. 2. Typical IPTV architecture

approach described above effectively uses virtualization to achieve this minimization of resource usage. Because VMs can be spawned quickly ([2]), the orchestration procedure exploits the prediction to do so in anticipation of an ICC load impulse. Furthermore, it causes the VoD VMs to serve existing sessions at a faster rate prior to the onset of the ICC load, and then quiesce these VMs during the ICC spike. This is show in Figure 1.

II. A Typical IPTV Architecture

Let us examine the architecture of an IPTV infrastructure that has been deployed nationwide to support both live broadcast TV and Video-On-Demand (VOD) delivery. Figure 2 shows a typical IPTV architecture for deploying LiveTV and VoD services.

Typically LiveTV is multicast from distribution servers (D-servers) using IP Multicast (PIM-SSM in this case [3]), with one group per TV channel. However, supporting instant channel change (ICC) [1] requires a unicast stream to each STB for a short period of time. When a user tunes to a TV channel by joining a particular multicast group for that channel, the content has to be delivered to the STB and fill its playout buffer. Since we wish to keep the switching latency small to satisfy the desired user experience, this unicast stream has to be delivered at a higher rate than the streaming rate of the video. The playout point may in fact comprise a few seconds of video to be played out to accommodate all the variability observed in the video delivery system. Although there are several solutions to mitigate the switching latency and realize instant channel change for the user [1], [4], in this paper we consider the approach by vendors of deployed equipment that delivers the video for the new channel at an accelerated rate using a unicast stream from the server at the VHO. The playout buffer is thereby filled quickly, and keeps switching latency small. Once the playout buffer is filled up to the playout point, the STB joins the multicast group to receive content, without continuing to impose a load on an individual basis on the VHO server complex. Other than ICC, broadcast of Live TV has a significant multicast component and supporting LiveTV multicast requires, relatively, smaller amounts of VHO server resources. The primary scalability challenge for LiveTV is the server resources for ICC. ICC adds a transient demand proportional to the number of users concurrently initiating a channel change event. We have observed that there is a dramatic burst load that is placed on the D-servers by correlated channel change requests from consumers. This results in very large peaks occuring on every half-hour and hour boundaries. The peak to average ratio can be multiple orders of magnitude and these peaks last for a few seconds (of the order of 10-30 seconds, required to fill the playout buffer). This means that the D-servers are over-provisioned for the majority of the time. It would be highly desirable to smooth out this very "peaky" load on the D-servers, and share these servers at non-peak instants to provide other services. Video-on-Demand (VoD) on the other hand has a relatively small number of servers currently, but is expected to grow. Each VoD request is satisfied with a unicast stream. While VoD servers also experience varying load from the subscriber population, the load is somewhat steady over these time scales of a half-hour or one-hour intervals. The receiving set-top boxes (STBs) also have sufficient storage to be able to buffer several seconds, or even minutes of video in high-speed memory (of course, there is a disk that could also store hours of video, but it is not used as part of the playout buffer to the decoder and display and is not considered here). By adapting the VoD delivery rate from the servers and pre-buffering the VoD content on the STB, we can easily adapt to the potential (un)availability of the server for short time intervals. This enables us to consider re-using the VoD server for meeting the LiveTV ICC request load.

Thus, there is a natural sharing possible of both LiveTV services and VoD on a common server complex. Our project of consolidation through virtualization is particularly compelling for a service provider because we do not necessarily need to modify the source code for the D servers and VoD servers. We can view a VM as a container for each of these server implementations, and thus rapidly prototype a shared server complex that integrates both these services. We think additional servers that are used for ad-insertion and other capabilities may also be integrated together, thus bringing down the cost, footprint and power consumption of the server complex in a VHO.

III. Computation of Resources for services with deadline constraint

There have been multiple efforts in the past on modeling to estimate the resource requirements for serving arrivals within a certain delay constraint, especially in the context of voice processing, including VoIP assuming Poisson processes [5]. We are different from these efforts since our results apply for any general arrival process. Our optimization algorithm computes the minimum number of servers needed based on the sum of the peaks of the composite workload. We also examine the amount of server resources required as the deadline constraint is varied. We then examine the benefit of multiplexing diverse services on a common infrastructure, and show how by dynamically adjusting the resources provided to a particular service while delaying the other service can bring significant savings in resource requirements in comparison to provisioning resources for each of the services independently. This would reflect the amount of 'cloud resources' required with multiple real-time services in the cloud infrastructure.

A. Single Service

Suppose there is a sequence of time instants during which a number of requests arrive to an incoming queue, denoted by $c(n)$ for $n = 1, \cdots N$. Each request has a deadline of m time units to be completely served after its arrival. In this case, a question arises as to what is the server capacity needed so that all the requests arriving at each of the n time instants are served with no request missing its deadline.

When $m = 0$, no request can have any delay and thus the number of servers that are needed is exactly the peak of $c(n)$ ($\max_{1\le n\le N} c(n)$). We find the trade-off between the number of servers needed and the deadline constraint tolerable. In addition we will assume that all the requests arriving within a time interval N have to be served within the same time interval.

The following theorem gives the number of servers needed to serve the requests within their deadline.

Theorem 1: Suppose that the incoming arrivals to a queue at time i is c_i, $1 \le i \le N$. Each of the request arriving at time i has a deadline of $\min(i + m, N)$. In this case, the number of servers given by,

$$S = \left\lceil \max\left\{ \max_{1\le i\le i+j\le N-m} \frac{\sum_{n=i}^{i+j} c(n)}{m+j+1}, \max_{0\le k<N} \frac{\sum_{j=0}^{k} c(N-j)}{k+1} \right\} \right\rceil, \quad (1)$$

is necessary and sufficient to serve all the incoming requests.

In case there is no restriction on all the requests being served by time N, this is equivalent to lengthening the process $c(i)$ to a length $N + m$ arrival process where $c(i) = 0$ for $i > N$. This gives us the following corollary.

Corollary 2: Suppose that the incoming arrivals to a queue are time i is c_i, $1 \le i \le N$ and no request is arriving for times $i > N$. Each of the request arriving at time i has a deadline of $i + m$. In this case, the number of servers given by,

$$S = \left\lceil \max_{1\le i\le i+j\le N} \frac{\sum_{n=i}^{i+j} c(n)}{m+j+1} \right\rceil, \quad (2)$$

is necessary and sufficient to serve all the incoming requests.

Corollary 3: When the service cannot have any delay, (or $m = 0$), the number of servers that is necessary and sufficient is given by $\max_{1\le n\le N} c(n)$.

We will prove Theorem 1 in the remaining part of the section.

We will first show the necessity of S servers. There are $c(j)$ requests arriving at time j and at most S requests can leave the queue. If $\sum_{n=i}^{i+j} c(n) > (m + j + 1)S$, the number of requests arriving from times $[i, i + j]$ cannot have departed in $m + j + 1$ time starting from time i. Thus, some request will miss the deadline. So, $\sum_{n=i}^{i+j} c(n) \le (m + j + 1)S$ for all $i + j \le N - m$. Further, if $\sum_{j=0}^{k} c(N - j) > (k + 1)S$, the requests arriving in last $k + 1$ time would not have gone out of the queue. Thus, $\sum_{j=0}^{k} c(N - j) \le (k + 1)S$ for all $k < N$. This proves that the expression of S given Theorem 1 are necessary.

We will now prove that the number of servers given in Theorem 1 are sufficient. For the achievability, we use a first-in-first-out (FIFO) strategy for servicing the queue. We serve the first S packets in the queue at each time based on FIFO strategy if there are more than S packets waiting in the queue. If there are less than S packets in the queue, we serve all the requests. We will show that with S given as in Theorem 1 and using this strategy, no request will miss the deadline.

Consider a time instant i. Suppose that the last time before i that the queue became empty be $j - 1$ (There exist such point since the queue was empty at 0 and hence this point would be last if there was nothing else in between). If $i < j + m$, then the packets that have arrived from j to i have not missed deadline yet. If $i \ge j + m$, the packets that should have departed from time j to i should be at-least $\sum_{n=j}^{i-m} c(n)$ and since this is $\le (m + 1 + i - m - j)S = (i - j + 1)S$, these packets would have departed. So, no request from time j to i has missed deadline. This is true for all i, j that have deadline m time units away.

But, after the last time $j - 1$ when the queue becomes empty, we also need to see if all the requests have been served by time N since deadline for some packets here would be more stringent. Let $j - 1$ be the last time instance when the queue becomes empty. Then, from that point, number of packets that entered the queue are $\sum_{n=j}^{N} c(n)$. This is $\le (N - j + 1)S$ which are packets that can depart from time j to time N. Thus, there are no packets remaining to be served after time N.

B. Extension to more services

In this subsection, we will extend the result in Theorem 1 to more than one services. Let there be k services $c_1(i), \cdots c_k(i)$ for $1 \le i \le N$. Each of these services have a deadline associated for the requests, service c_j with deadline constraint m_j. In this case, the number of servers needed are given in the following theorem.

Theorem 4: Suppose that there are k arrival processes $c_j(i)$ for $1 \le j \le k$ and $1 \le i \le N$ to a queue at time i. Request $c_j(i)$ arriving at time i has a deadline of $\min(i + m_j, N)$. In this case, the number of servers given by (3) at the top of next page, is necessary and sufficient to serve all the incoming requests.

In case there is no restriction on all the requests being served by time N, this is equivalent to lengthening each incoming processes to a length $N + \max(m_1, \cdots m_k)$ arrival process where $c_j(i) = 0$ for $i > N$. This gives us the following corollary.

Corollary 5: Suppose that there are k arrival processes $c_j(i)$ for $1 \le j \le k$ and $1 \le i \le N$ to a queue at time i and no request is arriving for times $i > N$. Request $c_j(i)$ arriving at time i has a deadline of $i + m_j$. In this case, the number of servers given by,

$$S = \left\lceil \max\left\{ \max_{1\le i\le i+t\le N, t\ge \min(m_1,\cdots m_k)} \frac{\sum_{j=1}^{k}\sum_{n=i}^{i+t-m_j} c_j(n)}{t+1}, \max_{0\le l<N} \frac{\sum_{j=1}^{k}\sum_{i=0}^{l} c_j(N-i)}{l+1} \right\}\right\rceil, \quad (3)$$

$$S = \left\lceil \max_{1\le i\le i+t\le N, t\ge \min(m_1,\cdots m_k)} \frac{\sum_{j=1}^{k}\sum_{n=i}^{i+t-m_j} c_j(n)}{t+1} \right\rceil, \quad (4)$$

is necessary and sufficient to serve all the incoming requests.

Corollary 6: When none of the services can have any delay, (or $m_j = 0$), the number of servers that is necessary and sufficient is given by $\max_{1\le n\le N}\sum_{j=1}^{k} c_j(n)$.

The proof of necessity follows along the same lines as the proof of theorem 1. For the sufficiency, we use the strategy of Earliest Deadline Scheduling rather than FIFO which says that sort the requests by deadline and if there are less than S requests, serve all and otherwise serve the first S requests. Similar steps prove that none of the request misses deadline with this strategy and thus a detailed proof is omitted.

IV. Cloud Architecture for IPTV

Figure 3 shows a cloud-based architecture for providing on-demand services. Each service has a dynamic pool of resources, including computing, network, and storage, which are allocated from cloud providers.

For each service, we first establish a workload model, that predicts the volume of incoming requests over time and (thus the resource needed at a given point in time to satisfy these requirements). This can be based on historical data analysis, external event hints, etc. In the context of IPTV, apart from the regular diurnal pattern that exists for all services, LiveTV ICC has a large number of correlated requests arriving periodically. Second, our architecture allows each service to expose a set of control mechanisms for varying the resource requirements without sacrificing service quality. Virtualization enables many of these control mechanisms. For example, after speeding up VOD content delivery, we can simply pause the VOD-related VMs, and dynamically allocate VMs [2] to handle the LiveTV ICC workload.

The core of our architecture is a service orchestrator that takes the individual workload models of all the services as input. Effectively, the orchestrator acts as an overseer that 1) understands the resource requirements of each service, and 2) decides on the adaptation methods to reduce the overall resource consumption. We plan to address this as an optimization problem. In particular, the service orchestrator divides the continuous time domain into bins that start on $T_0, T_1, \ldots$. At the beginning of a time bin T_i, the orchestrator first executes scheduled operations for that bin, such that the allocated resources for each service are updated. Based on the most recent workload model prediction, the orchestrator then adds or modifies scheduled operations on T_j $(j > i)$.

V. Related Work

There are mainly two threads of related work, namely cloud computing and scheduling policies. Cloud computing has recently changed the landscape of Internet based computing, whereby a shared pool of configurable computing resources (networks, servers, storage) can be rapidly provisioned and released to support multiple services within the same infrastructure [6]. Due to its nature of serving computationally intensive applications, cloud infrastructure is particularly suitable for content delivery applications. Typically LiveTV and VoD services are operated using dedicated servers [4], while this paper considers the option of operating multiple services by careful rebalancing of resources in real time within the same cloud infrastructure.

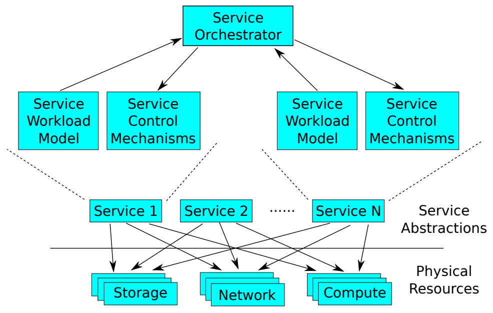

Fig. 3. Cloud IPTV architecture

VI. Experiments

We set up a series of experiments to see the effect of varying firstly, the ICC durations and secondly, the VoD delay tolerance on the total number of servers needed to accommodate the combined workload. All figures include a characteristic diurnal VoD time series (in pink) and a LiveTV ICC time series (in blue). Based on these two time series, the optimization algorithm described in section III computes the minimum number of concurrent sessions that need to be accommodated for the combined workload. The legends in each plot indicate the duration that each VoD session can be delayed by. Figure VI shows the superposition of the number of VoD sessions with a periodic LiveTV ICC session. The duration or the pulse width of the ICC session is 15 seconds (i.e. all ICC activity come in a burst and lasts for 15 seconds). We now compute the total number of concurrent sessions that the server needs to accommodate by delaying each VoD session from 1 second to 30 seconds in steps of 5 seconds. It is observed that as VoD sessions tolerate more delay the total number of sessions needed reduce to the point (15 sec delay) at which all ICC activity can be accommodated with the same number of servers that are provisioned for VoD. On the other hand, if the VoD service can afford only 1 second delay, the total number of sessions that need to be accommodated is roughly double (LiveTV ICC peak in red + VoD peak (blue)).

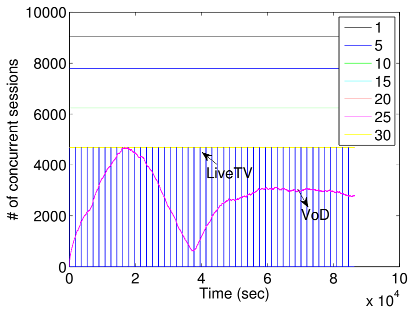

Fig. 4. Total # sessions needed with a 15 sec ICC pulse width - Synthetic trace

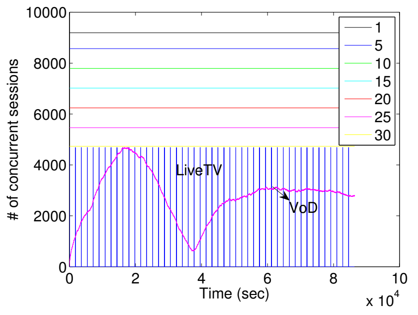

Fig. 5. Total # sessions that needed with a 30 sec ICC pulse width - Synthetic trace

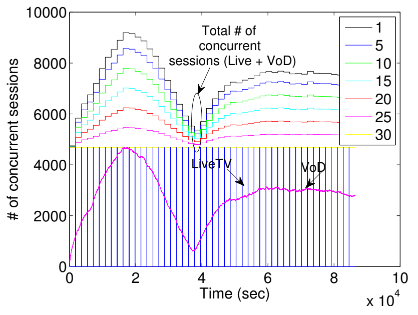

Fig. 6. Algorithm computed every 30 min, # concurrent sessions needed with a 30 sec hold time for ICC - Synthetic trace

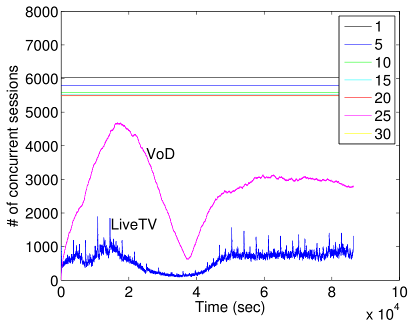

Fig. 7. # concurrent sessions needed with a 15 sec hold time for ICC - Operational trace

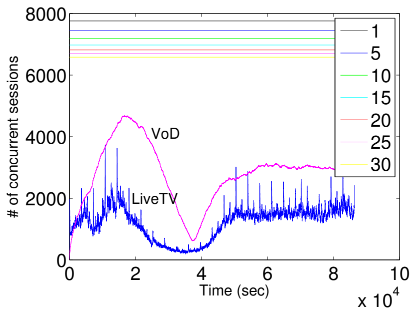

Fig. 8. # concurrent sessions needed with a 30 sec hold time for ICC - Operational trace

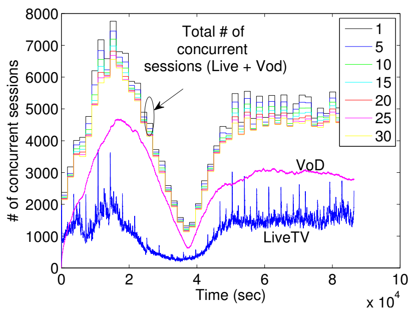

Fig. 9. Algorithm computed every 30 min, # concurrent sessions needed with a 30 sec hold time for ICC - Operational trace

Figure VI shows a similar effect, the only difference here is that the Live TV ICC pulse width is now 30 seconds.

Figure 7 and 8 shows the total number of concurrent sessions needed to accommodate the combined workload of VoD and LiveTV ICC requests. The traces are obtained from an operational IPTV environment in a relatively large VHO. We note that as VoD requests are delayed up to 30 seconds the total server bandwidth reduce by about 17.5%. In all there is a 21.67% saving in server bandwidth compared to sum of the peaks and 17.5% saving compared to peak of the sum with a 30 sec wait time allowed for VoD.

In the previous experiments we computed the minimum number of concurrent sessions that need to be supported based on observations over the entire day. Figure 9 shows the minimum number of concurrent sessions needed based on optimizing every half hour. Note that the peak number of sessions still coincide with Figure 8. However, as the load reduces, the number of concurrent sessions that need to be supported also reduce, thus tracking the diurnal pattern. Figure 6 shows similar observations, however, the LiveTV ICC trace is synthetically generated that peaks for 30 seconds every half hour.

VII. Conclusions

We investigate how an IPTV service can leverage cloud computing to schedule resources in an optimal manner. We provide an analysis that computes the minimum number of servers needed to accommodate a combination of IPTV services, namely VoD session and Live TV instant channel change bursts. By anticipating the LiveTV ICC bursts that occur every half hour we can speed up delivery of VoD content by prefilling the set top box buffer. This helps us to dynamically reposition the VoD servers for accommodating the LiveTV bursts that typically last for 15 to 30 seconds at most. Our results show that anticipating and thereby delaying VoD requests gives significant resource savings.

References

[1] D. Banodkar, K. K. Ramakrishnan, S. Kalyanaraman, A. Gerber, and O. Spatscheck, “Multicast instant channel change in IPTV system,” January 2008, proceedings of IEEE COMSWARE.

[2] H. A. Lagar-Cavilla, J. A. Whitney, A. Scannell, R. B. P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, “SnowFlock: Virtual Machine Cloning as a First Class Cloud Primitive,” *ACM Transactions on Computer Systems (TOCS)*.

[3] K. K. Ramakrishnan, R. Doverspike, G. Li, K. Oikonomou, and D. Wang, “IP Backbone Design for Multimedia Distribution: Architecture and Performance,” May 2007, proceedings of IEEE INFOCOM.

[4] “Microsoft tv: Iptv edition,” http://www.microsoft.com/tv/IPTVEdition.mspx.

[5] G. Ramamurthy and B. Sengupta, “Delay analysis of a packet voice multiplexer by the ΣDi/D/1 Queue,” July 1991, proceedings of IEEE Transacations on Communications.

[6] R. Urgaonkar, U. Kozat, K. Igarashi, and M. J. Neely, “Dynamic resource allocation and power management in virtualized data centers,” March 2010, proceedings of IEEE IFIP NOMS.