Optimizing Cloud Resources for Delivering IPTV Services through Virtualization

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

***Abstract*—Virtualized cloud-based services can take advantage of statistical multiplexing across applications to yield significant cost savings to the operator. However, achieving similar benefits with real-time services can be a challenge. In this paper, we seek to lower a provider's costs of real-time IPTV services through a virtualized IPTV architecture and through intelligent time-shifting of service delivery. We take advantage of the differences in the deadlines associated with Live TV versus Video-on-Demand (VoD) to effectively multiplex these services.**

We provide a generalized framework for computing the amount of resources needed to support multiple services, without missing the deadline for any service. We construct the problem as an optimization formulation that uses a generic cost function. We consider multiple forms for the cost function (e.g., maximum, convex and concave functions) to reflect the different pricing options. The solution to this formulation gives the number of servers needed at different time instants to support these services. We implement a simple mechanism for time-shifting scheduled jobs in a simulator and study the reduction in server load using real traces from an operational IPTV network. Our results show that we are able to reduce the load by $\sim 24\%$ (compared to a possible $\sim 31\%$). We also show that there are interesting open problems in designing mechanisms that allow time-shifting of load in such environments.

I. Introduction

As IP-based video delivery becomes more popular, the demands placed upon the service provider's resources have dramatically increased. Service providers typically provision for the peak demands of each service across the subscriber population. However, provisioning for peak demands leaves resources under utilized at all other periods. This is particularly evident with Instant Channel Change (ICC) requests in IPTV.

In IPTV, Live TV is typically multicast from servers using IP Multicast, with one group per TV channel. Video-on-Demand (VoD) is also supported by the service provider, with each request being served by a server using a unicast stream. When users change channels while watching live TV, we need to provide additional functionality to so that the channel change takes effect quickly. For each channel change, the user has to join the multicast group associated with the channel, and wait for enough data to be buffered before the video is displayed; this can take some time. As a result, there have been many attempts to support instant channel change by mitigating the user perceived channel switching latency [1], [2]. With the typical ICC implemented on IPTV systems, the content is delivered at an accelerated rate using a unicast stream from the server. The playout buffer is filled quickly, and thus keeps switching latency small. Once the playout buffer is filled up to the playout point, the set top box reverts back to receiving the multicast stream.

ICC adds a demand that is proportional to the number of users concurrently initiating a channel change event [1]. Operational data shows that there is a dramatic burst load placed on servers by correlated channel change requests from consumers (refer Figure 1). This results in large peaks occuring on every half-hour and hour boundaries and is often significant in terms of both bandwidth and server I/O capacity. Currently, this demand is served by a large number of servers grouped in a data center for serving individual channels, and are scaled up as the number of subscribers increases. However this demand is transient and typically only lasts several seconds, possibly upto a couple of minutes. As a result, majority of the servers dedicated to live TV sit idle outside the burst period.

Our goal in this paper is to take advantage of the difference in workloads of the different IPTV services to better utilize the deployed servers. For example, while ICC workload is very bursty with a large peak to average ratio, VoD has a relatively steady load and imposes "not so stringent" delay bounds. More importantly, it offers opportunities for the service provider to deliver the VoD content in anticipation and potentially out-of-order, taking advantage of the buffering available at the receivers. We seek to minimize the resource requirements for supporting the service by taking advantage of statistical multiplexing across the different services - in the sense, we seek to satisfy the peak of the sum of the demands of the services, rather than the sum of the peak demand of each service when they are handled independently. Virtualization offers us the ability to share the server resources across these services.

In this paper, we aim a) to use a cloud computing infrastructure with virtualization to dynamically shift the resources in real time to handle the ICC workload, b) to be able to anticipate the change in the workload ahead of time and preload VoD content on STBs, thereby facilitate the shifting of resources from VoD to ICC during the bursts and c) solve a general cost optimization problem formulation without having to meticulously model each and every parameter setting in a data center to facilitate this resource shift.

In a virtualized environment, ICC is managed by a set of VMs (typically, a few VMs will be used to serve a popular channel). Other VMs would be created to handle VoD requests. With the ability to spawn VMs quickly [3], we believe we can shift servers (VMs) from VoD to handle the ICC demand in a matter of a few seconds. Note that by being able to predict

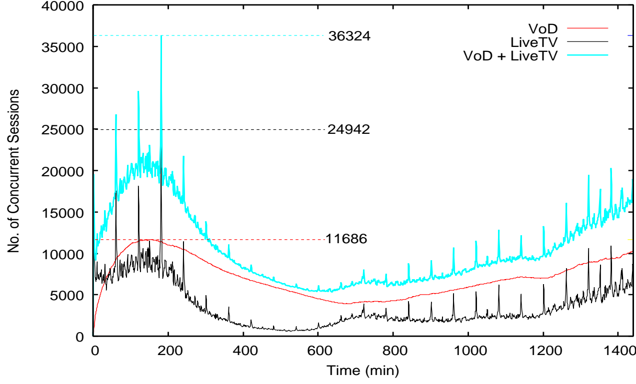

Fig. 1. LiveTV ICC and VoD concurrent sessions vs time, ICC bursts seen every half hour

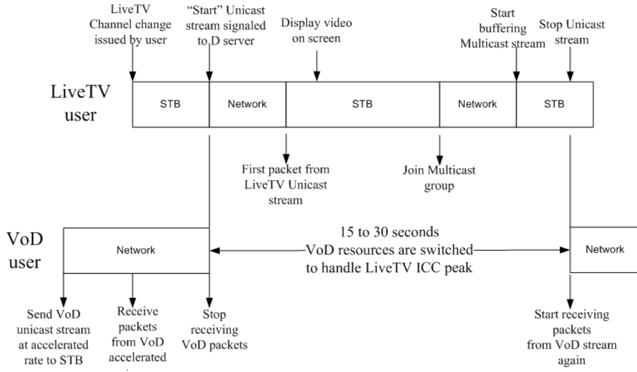

Fig. 2. LiveTV ICC and VoD packet buffering timeline

the ICC bursts (channel change behavior can be predicted from historic logs as a result of LiveTV show timings. The channel changes usually occurs every half hour (see Figure 1)). Figure 1 also shows the corresponding VoD load and the aggregate load for both services together. In anticipation of the ICC load, we seek to accelerate delivery of VoD content (for example, for a small number of minutes of playout time) to the users' STBs and shift the VoD demand away from the ICC burst interval (see Figure 2). This will also ensure that VoD users will not notice any impairment in their delivered quality of service (e.g. frozen frames etc.) as the playout can be from the local STB cache.

In preliminary work on this topic [4], we analyzed the maximum number of servers that are needed to service jobs with a strict deadline contraint. We also assume non-causal information (i.e., all deadlines are known a priori) of the jobs arriving at each instant. In this paper, we consider a generalized cost function for the servers. The cost of servers in this model can be a function of time, load, etc. Our goal is to find the number of servers at each time instant by minimizing this generalized cost function while at the same time satisfying all the deadline constraints.

We identify the *sever-capacity* region formed by servers at each time instant such that all the jobs arriving meet their deadlines, which is defined as: the region such that for any server tuple with integer entries inside this region, all deadlines can be met and for any server tuple with integer entries outside this region, there will be at least one request that misses the deadline. We also show that for any server tuple with integer entries inside the server-capacity region, an earliest deadline first (EDF) strategy can be used to serve all requests without missing their deadlines. This is an extension of previous results in the literature where the number of servers are fixed at all times [5]. The server-capacity region is formed by linear constraints, and thus this region is a polytope.

Having identified the server-capacity region in all its generality, we consider the cost function to be one of several possibilities: a separable concave function, a separable convex function, or a maximum function. We note that even though the functions are concave/convex, the feasible set of server tuples is all integer tuples in the server-capacity region. This integer constraint makes the problem hard, in general. We show that for a piecewise linear separable convex function, an optimal strategy that minimizes the cost function can be easily described. Furthermore, this strategy only needs causal information of the jobs arriving at each time-instant. For any concave cost function, we show that the integer constraint can be relaxed since all the corner points of the server-capacity region (which is a polytope) have integer coordinates. Thus, well known concave programming techniques without integer constraints can be used to solve the problem [6]. Finally, for a maximum cost function, we seek to minimize the maximum number of servers used over the entire period. This paper finds a closed form expression for the optimal value for the maximum number of servers needed based on the non-causal information of the job arrival process.

We show two examples of the cost function for computing the number of servers in Section V namely, the maximum and piecewise linear convex cost functions. We set up a series of numerical simulations 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. Our findings indicate that potential server bandwidth savings of (20% - 25%) can be realized by anticipating the ICC load and thereby shifting/smoothing the VoD load ahead of the ICC burst. Finally, we show by means of a faithful simulator implementing both these services in Section VI, that a careful choice of a lookahead smoothing window can help to average the additional VoD load. Ultimately our approach only requires a server complex that is sized to meet the requirements of the ICC load, which has no deadline flexibility, and we can *almost completely mask* the need for any additional servers for dealing with the VoD load.

II. Related Work

There are mainly three threads of related work, namely cloud computing, scheduling with deadline constraints, and optimization. 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 [7]. 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 [2], while this paper considers the option of operating multiple services by careful rebalancing of resources in real time within the same cloud infrastructure.

Arrival of requests that have to be served by a certain deadline have been widely studied [8], [9]. For a given set of processors and incoming jobs characterized by arrival time and requirement to finish by certain deadline, EDF (Earliest Deadline First) schedules the jobs such that each job finishes by the deadline (if there are enough processors to serve) [10]. In this paper, there are multiple sets of services providing jobs. Each of these services send request for chunks with different deadlines. For a fixed number of processors, EDF is optimal schedule. In this paper, we find the region formed by server tuples so that all the chunks are serviced such that no chunk misses deadline.

Optimization theory is a mathematical technique for determining the most profitable or least disadvantageous choice out of a set of alternatives. Dynamic optimization is a subbranch of optimization theory that deals with optimizing the required control variables of a discrete time dynamic system. In this paper, we consider finite-horizon optimization where the optimal control parameters with finite look-ahead are to be found [11] [12]. More specifically, we know the arrival pattern of the IPTV and VoD requests with their deadlines in the future. We wish to find the number of servers to use at each time so as to minimize the cost function. In this paper, we consider different forms of cost functions. We derive closed form solutions where possible for various cost functions.

III. Optimization Framework

An IPTV service provider is typically involved in delivering multiple real time services, such as Live TV, VoD and in some cases, a network-based DVR service. Each service has a deadline for delivery, which may be slightly different, so that the playout buffer at the client does not under-run, resulting in a user-perceived impairment. In this section, we analyze the amount of resources required when multiple real time services with deadlines are deployed in a cloud infrastructure.

There have been multiple efforts in the past to analytically estimate the resource requirements for serving arriving requests which have a delay constraint. These have been studied especially in the context of voice, including delivering VoIP packets, and have generally assumed the arrival process is Poisson [13]. We first extend the analysis so that our results apply for any general arrival process and we also consider multiple services with different deadlines. Our optimization algorithm computes the number of servers needed at each time (the *server-tuple*) based on the composite workload of requests from these different services. The optimization goal is to minimize a cost function which depends on the server-tuple such that all the deadline constraints are satisfied. We also study the impact of relaxing the deadline constraint on the optimal cost. Subsequently, we quantify the benefit of multiplexing diverse services on a common infrastructure. We show that significant resource savings can be achieved by dynamically allocating resources across services, as compared to provisioning resources for each service independently. For example, this can be used to exploit the same server resources to deliver Live TV as well as VoD, where their deadlines can be different, and in the case of VoD we can prefetch content in the the STB buffer. Our analysis is applicable to the situation where 'cloud resources' (e.g., in the VHO) are dynamically allocated to a service by exploiting virtualization.

Formulation - Let $r_j(i)$ denote the number of class-j requests arriving at time instant i, $i \in \{1, 2, \ldots, T\}$, $j \in \{1, 2, \ldots, k\}$, where k denotes the number of service classes. Every class-j request has deadline d_j, which means that a class-j request arriving at time i must be served at time no later than $\min\{i + d_j, T\}$. In the special case where there is only one service class, the subscripts are dropped, so that the number of requests at time i is denoted $r(i)$ and the deadline is denoted by d. Let us suppose that s_i servers are used at time i. The cost of providing service over a time interval $1 \leq i \leq T$ is denoted by $C(s_1, s_2, \cdots, s_T)$. Our goal is to minimize $C(s_1, s_2, \cdots, s_T)$ over this time interval, while ensuring that all the deadlines are met. By definition, a *request-tuple* is obtained by arranging the numbers $r_j(i)$ into a vector of length kT. It is understood that server-tuples and request-tuples are vectors of non-negative integers.

Given a request-tuple, a server-tuple $(s_1, s_2, \ldots, s_T)$, $s_i \in \mathbb{Z}^+$ (where $\mathbb{Z}^+$ denotes the set of whole numbers) is said to be *achievable* if all requests can be served within their deadlines. The *server-capacity* region for a given request-tuple is defined to be a region where all the integer coordinate points in the region are achievable while none of the integer coordinate point outside the region is achievable.

Consider the one-class case. When $d = 0$ each request must be served at the instant it arrives and the number of servers needed at time i is at least $r(i)$. Thus the server-capacity region is given by $\{(s_1, s_2, \ldots, s_T) : \ s_i \geq r(i),\, i = 1, 2, \ldots, T\}$. This means that for any server-tuple $(s_1, s_2, \ldots, s_T)$ with $s_i \in \mathbb{Z}^+$ in the sever-capacity region, all the incoming requests will be satisfied and for any server-tuple $(s_1, s_2, \ldots, s_T)$ with $s_i \in \mathbb{Z}^+$ not in the region the deadline for at least one request will not be met.

The following theorem characterizes the server-capacity region for a given request-tuple. We provide an example of a server-capacity region in Figure 3.

Theorem 1. *Given that all service requests belong to the same class, the server-capacity region is the set of all server-tuples* $(s_1, s_2, \ldots, s_T)$ *which satisfy*

$$\sum_{n=i}^{i+t-d} r(n) \leq \sum_{n=i}^{i+t} s_n \quad \forall 1 \leq i \leq i+t \leq T, t \geq d, \tag{1}$$

$$\sum_{n=0}^{l} r(T-n) \leq \sum_{n=T-l}^{T} s_n \forall 0 \leq l \leq T. \tag{2}$$

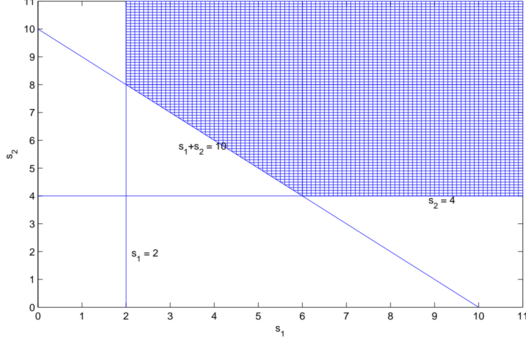

Fig. 3. Example server-capacity region for $s1 \geq 2$, $s2 \geq 4$, $s1 + s2 \geq 10$. An example setting is $T = 2$, $k = 2$, $r_1(1) = 2$, $r_1(2) = 4$, $r_2(1) = r_2(2) = 2$, $d_1 = 0$, $d_2 = 1$.

Equation (1) suggests that the total number of servers needed in time window i and $i+t$ has to be greater than or equal to the sum total of all arriving jobs in time window i and $i+t$ that have deadlines in the same window (ignoring the boundary condition that the jobs have to depart by time-instant T, which is the end of the time interval over which we estimate the required capacity). Equation (2) indicates a boundary condition that all jobs have to be completed and delivered by the time instant T. This theorem is a special case of the following theorem.

Theorem 2. *Suppose that there are k service-classes. The server-capacity region is the set of all server-tuples $(s_1, s_2, \ldots, s_T)$ which satisfy*

$$\sum_{j=1}^{k} \sum_{n=i}^{i+t-d_j} r_j(n) \leq \sum_{n=i}^{i+t} s_n \quad \forall 1 \leq i \leq i+t \leq T, \; t \geq \min(d_1, \cdots, d_K), \tag{3}$$

$$\sum_{j=1}^{k} \sum_{n=0}^{l} r_j(T-n) \leq \sum_{n=T-l}^{T} s_n \forall 0 \leq l \leq T. \tag{4}$$

Proof: **Converse:** We will first show the necessity for $(s_1, s_2, \cdots, s_T)$ servers. There are $\sum_{j=1}^{k} r_j(i)$ requests arriving at time i and at most s_i requests can depart at that time instant. The number of requests that have to depart in time-window $[i, i+t]$ has to be at-least $\sum_{j=1}^{k} \sum_{n=i}^{i+t-d_j} r_j(n)$. The maximum number of requests that can depart is $\sum_{n=i}^{i+t} s_n$. Thus, if $\sum_{j=1}^{k} \sum_{n=i}^{i+t-d_j} r_j(n) > \sum_{n=i}^{i+t} s_n$, there will be at least one request that would miss its deadline. Furthermore, for the boundary condition of all jobs arriving in the trailing time-window, if $\sum_{j=1}^{k} \sum_{i=0}^{l} r_j(T-i) > \sum_{i=T-l}^{T} s_i$, the requests arriving in last $l+1$ time instances would not have departed by time T. Thus, if the servers $(s_1, s_2, \cdots, s_T)$, $s_i \in \mathbb{Z}^+$ is outside the region given by (3)-(4), some job will miss its deadline. Thus, $(s_1, s_2, \cdots, s_T)$, $s_i \in \mathbb{Z}^+$, has to be inside the region given by Equations (3)-(4) for all the deadlines to be met.

Achievability: We will now prove that if the number of servers $(s_1, s_2, \cdots, s_T)$, $s_i \in \mathbb{Z}^+$, are in the region given by (3) and (4), all the requests will be served on or before their deadline. For achievability, we use an Earliest Deadline First (EDF) strategy for servicing the queue of requests. We serve the first s_i packets in the queue at time i based on EDF strategy, if there are more than s_i packets waiting in the queue. If there are less than s_i packets in the queue, obviously we will serve all the requests.

Next, we will show that if $(s_1, s_2, \cdots, s_T)$, $s_i \in \mathbb{Z}^+$, are in the region specified by (3) and (4), no request will miss its deadline. Consider a time instant $i < T$. Suppose that the last time instant prior to i that the queue became empty is $j-1$ (There exists such a point since the queue was empty at at least at time instant 0. So, time instant 0 would be last point if the queue was not empty at any subsequent point before i). If $i < j + \min(d_1, \cdots, d_K)$, then the requests that arrived from j to i have not missed their deadlines yet. If $i \geq j + \min(d_1, \cdots, d_K)$, the packets that should have departed from time j to i should be at least $\sum_{u=1}^{k} \sum_{n=j}^{i-d_u} r_u(n)$ and since this is $\leq \sum_{n=j}^{i} s_n$, these requests would have departed. Therefore, no request from time j to i has missed its deadline yet. Thus, no deadline has been missed till time $T-1$.

We now consider $i = T$. After the last time $j-1$ when the queue became empty, we need to examine if all the requests have been served by time T, since the deadline for some requests arriving in the time between j and T would be more stringent. Let $j-1$ be the last time instant when the queue last became empty. Then, from that point on, the number of requests that arrived is $\sum_{u=1}^{k} \sum_{v=j}^{T} r_u(v)$. This is $\leq \sum_{v=j}^{T} s_v$, which is the number of requests that were served from time j to time T. Thus, there are no requests remaining to be served after time T. This proves that if $(s_1, s_2, \cdots, s_T)$, $s_i \in \mathbb{Z}^+$, are in the region given by the (3) and (4), no request will miss its deadline.

■

Thus, our optimization problem reduces to minimizing $C(s_1, s_2, \cdots, s_T)$, such that Equations (3) and (4) are satisfied, and $s_i \in \mathbb{Z}^+$.

Note that the region given by Equations (3)-(4) can be represented as $T(T+1)/2$ constraints given by $\sum_{i=i_1}^{i_2} s_i \geq P(i_1, i_2)$ for $T \geq i_2 \geq i_1 \geq 0$, where $P(i_1, i_2)$ is the function fully characterized by the requests and the deadlines and can be derived from equations (3)-(4). Thus, the optimization problem is equivalent to minimizing $C(s_1, s_2, \cdots, s_T)$, such that $\sum_{i=i_1}^{i_2} s_i \geq P(i_1, i_2)$ for $T \geq i_2 \geq i_1 \geq 0$ and $s_i \in \mathbb{Z}^+$.

IV. Impact of Cost Functions on Server Requirements

In this section, we consider various cost functions $C(s_1, s_2, \cdots, s_T)$, evaluate the optimal server resources needed, and study the impact of each cost function on the optimal solution.

A. Cost functions

We investigate linear, convex and concave functions (See Figure 4). With convex functions, the cost increases slowly initially and subsequently grows faster. For concave functions,

the cost increases quickly initially and then flattens out, indicating a point of diminishing unit costs (e.g., slab or tiered pricing). Minimizing a convex cost function results in averaging the number of servers (i.e., the tendency is to service requests equally throughout their deadlines so as to smooth out the requirements of the number of servers needed to serve all the requests). Minimizing a concave cost function results in finding the extremal points away from the maximum (as shown in the example below) to reduce cost. This may result in the system holding back the requests until just prior to their deadline and serving them in a burst, to get the benefit of a lower unit cost because of the concave cost function (e.g., slab pricing). The concave optimization problem is thus optimally solved by finding boundary points in the server-capacity region of the solution space.

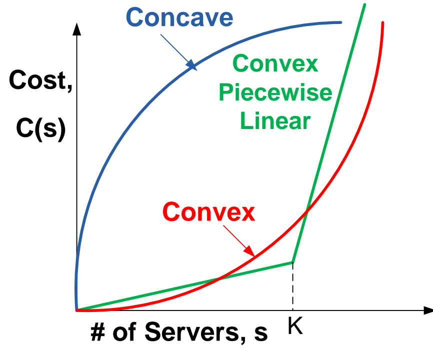

Fig. 4. Cost functions

We consider the following cost functions:

1) Linear Cost: $C(s_1, s_2, \cdots, s_T) = \sum_{i=1}^{T} s_i$. This models the case where we incur a cost that is proportional to the total number of servers needed across all times.

2) Convex Separable Cost: $C(s_1, s_2, \cdots, s_T) = \sum_{i=1}^{T} C(s_i)$, where $C(s_i)$ is a convex function. This models the case when a data center sees an increasing per unit cost as the number of servers required grows. We consider two examples of $C(s_i)$, the component cost function. The first is the exponential function, $C(s_i) = exp(s_i)$. The second is a piecewise linear function of the form $C(s_i) = s_i + c(s_i - K)^+$ where $c, K \geq 0$. This component cost function has per-server cost of unity when $s_i \leq K$, and per-server cost of $1 + c$ thereafter.

3) Concave Separable Cost: $C(s_1, s_2, \ldots, s_T) = \sum_{i=1}^{T} C(s_i)$, with component cost $C(s_i)$ a concave function. This may arise when the per-server cost diminishes as the number of servers grows.

4) Maximum Cost: $C(s_1, s_2, \cdots, s_T) = \max_{i=1}^{T} s_i$. This cost function penalizes the peak capacity that will be needed to serve the incoming sequence of requests.

B. Examples

In this subsection, we provide a few examples to illustrate the optimization framework. We consider both the case of a single service, and the case of two services.

Single service: Let us consider a single service, whose requests $r(i)$, arrive, each with a deadline of $d = 5$. Consider a total time period of $T = 100$. Consider as the arrival requests the sequence $[10, 0]$ repeated until time $T = 100$. The convex optimization algorithm will yield a requirement of $s_i = 5$ for all $1 \leq i \leq 100$ as the optimum number of servers. This is because the total number of 10 arrivals in a time span of 2 time units will have to be served prior to their deadline of $d = 5$, and can be done with 5 servers. Since the linear cost function is a special case of convex optimization, we get the same result.

The concave algorithm tends to move the servers needed to a corner point of the server-capacity region, which is a polytope formed by the various deadline constraints. In this case, this will result in optimal server requirement at each time to be [0, 0, 0, 0, 30] over a span of 5 time units. This server requirement will be repeated for each of the 5-time unit spans until $T = 100$. We note that the solutions given here are the optimal solutions in terms of cost. However, the optimal solution need not be unique. The cost function based on the maximum number of servers will also show that the number servers needed is at least 5.

It can be seen that the optimal solution for the concave cost function is peaked, and this is accomplished by increasing the number of servers at certain times and significantly reducing the requirement at other times. On the other hand, the convex cost function results in smooth optimal solutions obtained by spreading jobs out as evenly as possible. In our case, this is seen through the optimal allocation of 5 servers at each time instant. The maximum cost function results in a solution that smoothes out the peak requirement by deferring requests at peak times to times when there are comparatively less requests.

Multiple services: In this example, let us consider two services namely VoD and Live TV (whose service requirement is primarily to serve the ICC requests). Consider the case that the VoD has an arrival process $r_1(n)$ that is a sequence of arrivals at each of the first 6 time instants as: [10, 0, 10, 0, 10, 0]. This is is repeated. The ICC service requests also arrive as follows over the first 6 time instants as [4, 10, 4, 10, 4, 10]. This is also repeated. The deadlines are one time slot $d_1 = 1$ for VoD and $d_2 = 0$ for ICC (i.e., there is no deadline flexibility and it has to be served within that time slot). The convex algorithm would result in the optimal number of servers being $s_i = 12$. Thus, two of the requests of VoD in the odd time instants are served only in next time instant. Since the linear cost function is a special case of convex optimization, the above is an optimal solution even for the linear cost function. For a concave cost function, the two feasible corner points are having a number of servers equal to [14, 10, 14, 10, 14, 10] over the first 6 time instants which is repeated], and a number of servers equal to [4, 20, 4, 20, 4, 20] which is then repeated. Note that for any concave function, the allocation of a number of servers [4, 20, 4, 20, ...] will cost no more than the cost of [14, 10, 14, 10, ...]. Thus the sequence of allocations of [4, 20, 4, 20], that is repeated, is the optimal solution. This solution holds back the VoD that have to be served along with the peak in the arrival of ICC requests. Thus, the maximum server cost will result in the

minimum number of servers needed, 12.

C. Optimal Solutions

1) Linear Cost Function: One strategy for meeting this cost is to set $s_i = \sum_{j=1}^{k} r_j(i)$, which means that we serve all requests as they arrive.

2) Piecewise Linear Convex Cost Function: Consider any scheduling of the incoming requests which uses y_i server resources a time i. Suppose that we only serve $\min(y_i, K)$ of the requests and drop the remaining. The cost of using servers y_i at time i is given by total number of requests + c times the number of dropped requests. We know that the earliest deadline first strategy minimizes the number of dropped requests and hence the optimal strategy for the cost is as follows. Suppose that when we use earliest deadline first as the strategy with K as the number of servers, $\hat{s}_i$ be the number of requests served in time i and $\tilde{s}_i$ is the number of requests dropped (Note that $\tilde{s}_i = 0$ if $\hat{s}_i < K$.). Then $s_i = \hat{s}_i + \tilde{s}_i$ is an optimal solution.

3) Exponential Cost Function: This is a convex optimization problem with integer constraints, and is thus NP hard problem in general. We here provide an achievable solution based on convex primal-dual method.

Initialization: Let all $s_i = \sum_{j=1}^{k} r_j(i)$.
Repeat (n=1, ...):
1. $\lambda_{i_1,i_2} = (\lambda_{i_1,i_2} + \delta_n(P(i_1,i_2) - \sum_{i=i_1}^{i_2} s_i))^+$, where $\delta_n = 1/n^{3/5}$ for all $1 \le i_1 \le i_2 \le T$.
2. $s_i = (\lfloor \log(\sum_{i_1=1}^{i} \sum_{i_2=i}^{T} \lambda_{i_1,i_2}) \rfloor)^+$ for all $i = 1, \cdots, T$.
3. $s_i = s_i + \max_{i_2 \ge i}(P(i,i_2) - \sum_{j=i}^{i_2} s_j)^+$ for all $i = 1, \cdots, T$.

We initialize the algorithm by serving all the requests in the time they arrive. The first step changes the value of lagrange multipliers based on the linear constraints. The second step updates the servers based on the lagrangian solutions and rounds the servers below to integers. The third step increases the servers so that the solution at each step will be achievable and all the constraints are satisfied.

4) Concave Cost Function: This is a concave programming with linear and integer constraints.

For a concave minimization with linear constraints, the solution is one of the corner points of the polytope formed by the linear constraints [14]. We will show that minimization with linear and integer constraints is the same as minimization with just the linear constraints. Thus, the solution to the problem is one of the corner points of the polytope formed by the linear constraints.

Theorem 3. *Minimizing a concave function of $(s_1, s_2, \cdots, s_T)$ with linear constraints of the form $P(i_1,i_2) - \sum_{j=i_1}^{i_2} s_j \le 0$ for all $1 \le i_1 \le i_2 \le T$ and integer constraints $s_i \in \mathbb{Z}^+$ for $P(i_1,i_2) \in \mathbb{Z}^+$ is the same as minimizing the same concave function with linear constraints of the form $P(i_1,i_2) - \sum_{j=i_1}^{i_2} s_j \le 0$ for all $1 \le i_1 \le i_2 \le T$, and linear constraints of the form $s_i \ge 0$. Thus, the integer constraint requirement can be relaxed.*

Proof: We note that the first problem with linear and integer constraint gives a higher cost than the problem with only linear constraints because we are minimizing over a smaller region. If we prove that the optimal solution of second problem is a feasible solution of first, we prove that the second problem has a higher cost thus proving that the optimal cost for the two problems is the same.

To show that optimal solution of second is a feasible solution of first, we need to prove that the optimal solution of second problem has $s_i \in \mathbb{Z}^+$. Since the optimal solution of the second problem is one of the corner points of the polytope formed by linear constraints, it is enough to show that all the feasible corner points have $s_i \in \mathbb{Z}^+$.

There are $T(T+1)/2$ of the linear constraints $P(i_1,i_2) - \sum_{j=i_1}^{i_2} s_j \le 0$. The feasible corner points are a subset of the points formed by solution of any T of the conditions $P(i_1,i_2) - \sum_{j=i_1}^{i_2} s_j = 0$. We now find the solution of any T equations of the form $\sum_{j=i_1}^{i_2} s_j = P(i_1,i_2)$. This is equivalent to $As = p$ for A as a $T \times T$ binary matrix, $s = [s_1, \cdots s_T]$ and p as a $T \times 1$ vector. We note that due to the structure of A, this can be reduced by Gaussian elimination to get A in row-echelon form maintaining the binary structure of A and maintaining p as an integer vector. At each step, choose the row with minimum number of entry 1 to subtract from all other rows from where a leading 1 is to be removed. This will result in a binary matrix at each step in the reduction. In this row-echelon form, if there are no zero rows, the matrix is full rank. We start with last row which gives s_T as integer and remove column and keep going back thus giving us a feasible solution which has all coordinates as integers.

If there are zero rows in the row-echelon form, A is not full rank. If t rows are zero, we check if the last t elements in p are zero or not. If not, this system of equations do not give a feasible solution. If yes, this results in a t-dimensional plane of feasible solutions which means that t of the equations are redundant and can be removed. Thus, this condition do not give any corner point solution since we can remove these t constraints and always add other constraints from our set of inequations to get corner points on this plane. ∎

Thus for any concave function, the problem is a concave minimization problem with linear constraints. Efficient solutions for this class of problem have been widely studied, see for example [14], [15].

5) Maximum Cost Function: For this cost function, the optimal cost is given in the following theorem.

Theorem 4. *Suppose that there are k arrival processes $r_j(i)$ for $1 \le j \le k$ and $1 \le i \le T$ to a queue at time i. Request $r_j(i)$ arriving at time i has a deadline of $\min(i + d_j, T)$. In this case, the peak number of servers given by (5) at the top of next page, is necessary and sufficient to serve all the incoming requests.*

Proof: The necessity of these number of servers follow by Theorem 1, in the sense that if there are S servers at each point, S has to be at-least as in the statement of this

$$S = \left\lceil \max\left\{ \max_{1\le i\le i+t\le T, t\ge \min(d_1,\cdots d_k)} \frac{\sum_{j=1}^{k}\sum_{n=i}^{i+t-m_j} r_j(n)}{t+1}, \max_{0\le l<T} \frac{\sum_{j=1}^{k}\sum_{i=0}^{l} r_j(T-i)}{l+1} \right\} \right\rceil, \tag{5}$$

Theorem. Further, these number of peak servers are sufficient also follows from Theorem 1 by using S servers at each time and using earliest deadline first strategy. ∎

In case there is no restriction on all the requests being served by time T, this is equivalent to lengthening each incoming processes to a length $T+\max(d_1, \cdots d_k)$ arrival process where $r_j(i) = 0$ for $i > T$. This gives us the following corollary.

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

$$S = \left\lceil \max_{1\le i\le i+t\le T, t\ge \min(d_1,\cdots d_k)} \frac{\sum_{j=1}^{k}\sum_{n=i}^{i+t-m_j} r_j(n)}{t+1} \right\rceil, \tag{6}$$

is necessary and sufficient to serve all the incoming requests.

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

V. Numerical results

A. Maximum cost function

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 blue) and a LiveTV ICC time series (in red). Based on these two time series, the optimization algorithm described in section IV 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 5 shows the superposition of the number of VoD sessions with a periodic synthetic LiveTV ICC session. The duration or the pulse width of the ICC session is set to be 15 seconds (i.e. all ICC activity come in a burst and lasts for 15 seconds), and the peak of the pulse is set to be the peak of the VoD concurrent sessions for the whole day. We now compute the total number of concurrent sessions that the server needs to accommodate by delaying each VoD session from 1 second to 20 seconds in steps of 5 seconds. It is observed that as VoD sessions tolerate more delay the total number of servers 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 thus resulting in 50% savings in the server bandwidth. 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. Figure 6 shows a similar effect, the only difference here is that an operational trace is used for LiveTV. We note that as VoD requests are delayed up to 20 seconds the total server bandwidth reduce by about 28% as compared to serving LiveTV and VoD without any delay.

B. Convex piecewise linear cost function

Figures 7 and 8 show the number of servers needed when a piecewise linear cost function is used. We note how for different values of K allocates substantially different number of servers. Figure 7 simulates a synthetic LiveTV ICC 15 sec pulse width of amplitude ≈ 12000 and an operational VoD measurement trace. When $K = 12500$ the number of servers needed peak with every incoming ICC burst (spaced 30 mins apart) and then bottom out at the VoD envelope (in blue). If we use a smaller K, e.g. $K = 10000$, many chunks will miss their deadlines (especially during the peak hour) if total of K servers are used. Thus, a large number of chunks have to be served with a higher cost (red spikes of value 2.3×10^4 during the busy hour). The number of chunks that need to be served with a higher cost is larger when K is smaller. For any K which is at-least given in Equation 5, all the requests can be served at the lower cost and hence never are more than K servers needed. For any K smaller than that in Equation 5, there would be a chunk that misses deadline with K servers and hence there will be at-least one chunk that is served at higher cost. Lower the value of K, more jobs need to be served with higher cost. Figure 8 portrays a similar story for an operational LiveTV trace. With a smaller K, jobs are delayed to create larger overshoots (red spike of value 3.6×10^4).

VI. Simulation

To demonstrate the efficacy of our proposal in the realistic situation of supporting IPTV services, we implemented the adjustment mechanism in a custom event-driven simulator. We evaluated our approach using the VoD and ICC requests seen in a large-scale operational IPTV service. The user request traces were collected over 24 hours at a set of VHOs in one state for both VoD and ICC. In all, our data more than 18 million VoD and ICC requests in that time period. Our results show that we can get a substantial reduction ($\sim$ 24% for our traces) in peak server load by adjusting the deadlines of VoD requests in anticipation of ICC requests. We present the details of our experiments and the results in this section.

A. Experiment Setup

We used a custom event-driven simulator to perform our experiments. The simulator models the clients and the servers but abstracts, and represents by a simple link, the complex network that typically connects them. We view each video as

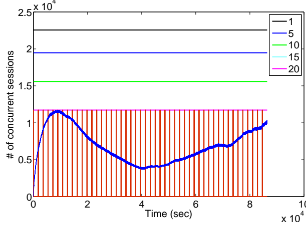

Fig. 5. Maximum cost: Total # sessions needed with a 15 sec ICC pulse width - Synthetic trace, deadline for VoD in secs shown in legend

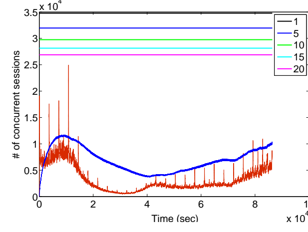

Fig. 6. Maximum cost: Total # sessions needed with a 15 sec ICC pulse width - Operational trace, deadline for VoD in secs shown in legend

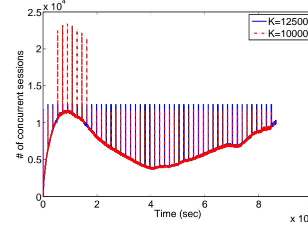

Fig. 7. Piecewise linear cost: Total # sessions needed with a 15 sec ICC pulse width - Synthetic trace, 15 sec deadline for VoD

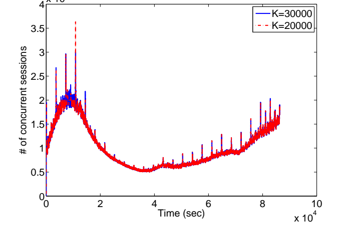

Fig. 8. Piecewise linear cost: Total # sessions needed with a 15 sec ICC pulse width - Operational trace, 15 sec deadline for VoD

comprising of chunks. This is similar to what most commercial streaming systems (e.g., HTTP Live Streaming, Smooth Streaming) employ today. When a client requests a video, it sends the requests to the server and in response identifies the chunks in video. We set each chunk to be up to 30 seconds long. Each client then requests N chunks in parallel from the server. In our experiments we nominally set $N = 25$. The server then schedules these requests according to their deadlines (i.e., the time by which the client should have that chunk) and transfers it before the deadline. Upon receiving a chunk, the client requests the next outstanding chunk.

We modelled ICC requests also as requests for video. We assumed that each ICC request results in 15 seconds of unicast video transferred to the client. As a result, each ICC request results in a request of one 15-second chunk of video that has to be delivered immediately.

As observed in the traces, there is a sudden burst in ICC requests every 30 minutes and lasts for a short duration. We call this the *ICC Burst Window*. We seek to minimize the load due to these ICC bursts by advancing the transfer of previously scheduled VoD requests. The process we adopt is depicted in Figure 9. Assuming that the ICC burst window lasts from time `t+s` to `t+b`, we start the adjustment process at an earlier point, at time `t`. In the window from `t` to `t+s`, which we call the *smoothing window*, we advance the jobs already scheduled in the ICC burst window. Thus these jobs are served prior to their deadline. However, in this process we make room for the increased number of ICC requests expected in the burst window. Note that we do not take any special action for the new VoD requests that arrive after the adjustment process has begun at time t, but before the end of the burst window. As a result, these requests will result in some continued VoD load during the burst window. One could however implement a more sophisticated scheme if the environment warrants it.

The reduction in load depends on multiple factors. First, we need to predict when the burst will occur. Next, we need to predict how long the burst's effect will last (i.e., burst window). We also need to predict how many VoD jobs scheduled in the burst window have to be moved. Finally, when we start the adjustment process, its time period for averaging the load (i.e., smoothing window) also plays a key role.

We study each of these effects in our experiments. We assume that our ICC spike occur at fixed 30 minute intervals. By default we assume that our burst window is one minute

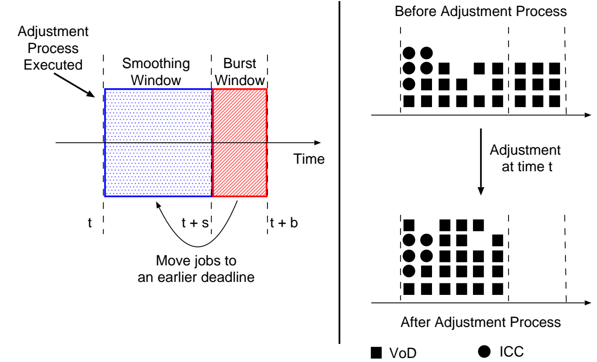

Fig. 9. Overview of the job adjustment process.

long, but we also experiment with two minute bursts. We typically assume that the adjustment starts 10 minutes before the burst window and that all the scheduled VoD jobs are randomly adjusted over the smoothing window. The smoothing window is also set at 10 minutes. Note that we ideally want to use a larger smoothing window than the burst window to spread out the load imposed by the moved jobs; otherwise we will experience a spike in the load during the smoothing window, thus negating the benefit of the procedure somewhat. Our primary metric is the number of concurrent streams that servers have to serve. We use this as our metric because it directly translates to how many servers are required to support the total request load. In particular, the *peak number* of concurrent streams is most relevant because it give the minimum number of servers required.

B. Establishing a baseline

In Figure 1, we separately plot the load due to VoD requests only, only ICC requests, and the combined load for both VoD and ICC requests. Figure 1 shows that if we did not do any adjustment, we would need to support a maximum of 36324 concurrent streams of VoD and ICC requests. If we only supported ICC requests, this goes down to 24942 streams. It reduces further to 11686 streams considering VoD alone. Recall that ICC requests result in 15 seconds of data transfer and are served immediately (the deadline is 0). Hence the best we can do is to go down to 24942 streams if we support both services, but are able to mask the VoD service completely (a 31.33% reduction). This gives us a baseline best case (lower bound) to compare the performance of our proposed adjustment mechanism.

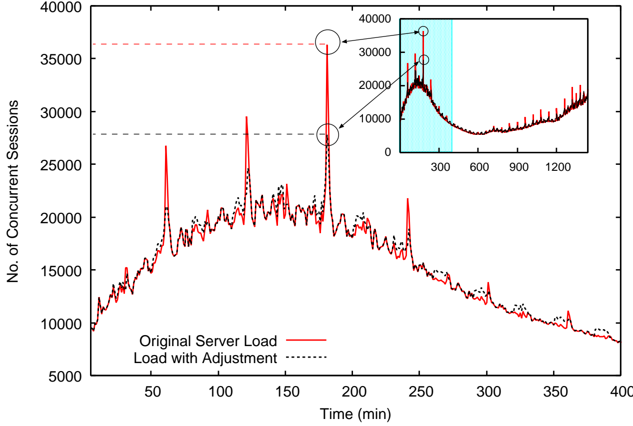

Fig. 10. Reduction in load due to job rescheduling.

C. Rescheduling jobs reduces load

For the main result, we show that rescheduling jobs to an earlier deadline is indeed posible and that it can result in a significant reduction in the aggregate load. We assume an ICC burst window of 1 minute and a smoothing window of 10 minutes. We also assume that all the VoD jobs prior to the burst window are moved to the beginning of the smoothing window. We plot the result in Figure 10.

The inset in Figure 10 shows the trends for the full day. To emphasize the performance during the peak period (e.g., the prime TV viewing time), we plot peak period for the day (the first 400 minutes, marked by the shaded region in the inset) during which the peak number of streams are served. The result of this experiment shows that with the adjustment, we are able to bring the peak number of concurrent streams down from 36324 streams to 27813 streams, a $\sim 24\%$ reduction. This number is very close to the load due to ICC requests alone, indicating that we have successfully moved all the VoD requests needed, making way for the ICC burst to be served in the burst window.

While this is a substantial reduction, it is lower than the possible 31% reduction. We attribute the lower gain to the way we exercise the time-shifting of the VoD load. Recall that the adjustment of serving VoD requests is done at the start of the smoothing window. However, any VoD requests that arrive after the adjustment is initiated cannot be rescheduled and results in load during the burst window as well. With a 10 minute smoothing window, we see quite a few of the new VoD requests after the adjustment is complete. To understand this interaction better, we study the effect of varying the size of the smoothing window next.

D. Effect of smoothing window size

The smoothing window size determines how the VoD load from the burst window is distributed. Choosing a small smoothing window results in more accurate determination of how many scheduled VoD jobs exist, but could result in a load spike within the smoothing window. On the other hand, a large smoothing window allows us the average the VoD load from the burst window better, but prevents the reschduling of many new VoD sessions that arrive subsequently. We quantify the effect of the smoothing window, while keeping the burst window at 2 minutes. VoD jobs are shifted from those two minutes. We vary the smoothing window from 2 minutes to 10 minutes and present the results in Figure 11.

Interestingly, we see that at the peak (around the 180 minute time marker), using a 5 minute smoothing window results in better performance than a 10 minute smoothing window (26979 vs. 27813 streams). We attribute the improvement to the ability to reschedule more VoD streams because of a smaller smoothing window. However it is not as simple as just employing a smaller window. When we reduce the smoothing window further, to 2 minutes, the load is consistenly higher than the other windows. Even more importantly, the 10 minute smoothing window consistently outperforms the others outside the peak viewing period (e.g., see at 300 minutes). This is because at the peak period, the number of ICC requests is significantly higher than the VoD requests. Hence moving as many VoD requests as possible is important. At other times, the number of VoD requests are higher. Hence moving all the VoD requests to an earlier deadline increases the load at that time. This is significant; it tells us that we need a more sophisticated approach to predicting the load in a burst and in choosing the size of the smoothing window.

E. Effect of Burst Window

Understanding the burst window is important as it tells us how long the burst is going to last. We study the effect of the size of the burst window by changing the burst window from 1 minute to 2 minutes, while keeping the smoothing window fixed at 10 minutes. We present the result in Figure 12. Interestingly, we see that the size of the burst window has only a small role to play during the peak. This is because the majority of the load during the peak comes from ICC requests and the new VoD sessions. However we see that outside the peak interval, using a smaller burst window results in lower load. This again can be attributed to the fact that the load in these periods is primarily due to VoD and moving more jobs (like we do with the 2 minute burst windows) is counter-productive. Finally, we see sharp reductions in load after the burst window of 2 minutes, but not with a burst window of 1 minute. This is again because we have move many more VoD jobs than necessary.

F. Probabilistically moving jobs

The burst window tells us the interval from which we need to move the VoD jobs, and the smoothing window gives us the duration over which we may schedule them. However, we also need to know *how many* jobs to move. To capture this, we probabilistically moved jobs to the smoothing window. We set the smoothing window at 10 minutes and the burst window at 2 minutes but varied the probability p of moving a job from 0.25 to 1.0 and plot the result in Figure 13. We note some interesting behavior. First, during the peak (marked with '1'), we see that *increasing* the probability of moving jobs *decreases* the number of concurrent streams. However, at other

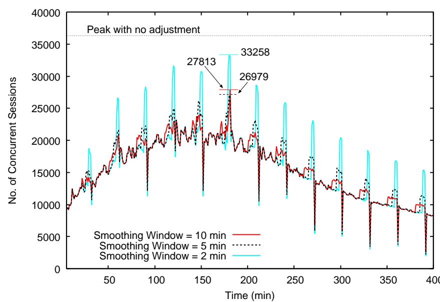

Fig. 11. Effect of smoothing window.

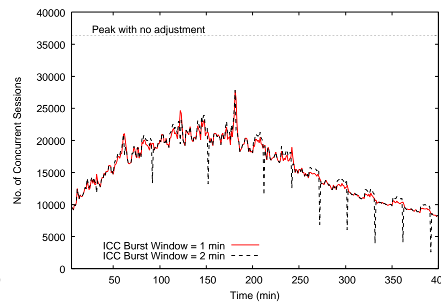

Fig. 12. Effect of burst window.

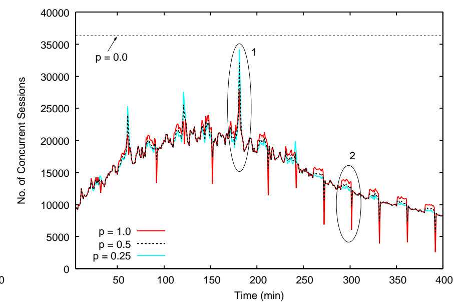

Fig. 13. Probability of moving jobs vs. load.

times (marked with '2'), *decreasing* the probability *decreases* the concurrent streams. This result clearly shows that we need a smarter way of figuring out how many jobs to move for this procedure to be applicable in a general.

G. Results Summary

In this section we presented results from a simple adjustment mechanism we implemented. Our results show that even our simple mechanism is able to give significant reductions in load. However, there is still room for improvement. We showed that the load reduction is dependent on the duration of the adjustment (burst window), the number of jobs moved and the period over which they are averaged (the smoothing window). Our results show that a particular value for each of these parameters is not the best across the board; instead the value chosen depends on the relative load of each of the services being adjusted. We believe that mechanisms to predict this relative load of each service and dynamically choose values for the parameters based on this prediction can yield further improvements. Designing such mechanisms is an opportunity for interesting future work.

VII. Conclusions

We studied how IPTV service providers can leverage a virtualized cloud infrastructure and intelligent time-shifting of load to better utilize deployed resources. Using Instant Channel Change and VoD delivery as examples, we showed that we can take advantage of the difference in workloads of IPTV services to schedule them appropriately on virtualized infrastuctures. By anticipating the LiveTV ICC bursts that occur every half hour we can speed up delivery of VoD content before these bursts by prefilling the set top box buffer. This helps us to dynamically reposition the VoD servers to accomodate ICC bursts that typically last for a very short time.

Our paper provided generalized framework for computing the amount of resources needed to support multiple services with deadlines. We formulated the problem as a general optimization problem and computed the number of servers required according to a generic cost function. We considered multiple forms for the cost function (e.g., min-max, convex and concave) and solved for the optimal number of servers that are required to support these services without missing any deadlines.

We implemented a simple time-shifting strategy and evaluated it using traces from an operational system. Our results show that anticipating ICC bursts and time-shifting VoD load gives significant resource savings (as much as 24%). We also studied the different parameters that affect the result and show that their ideal values vary over time and depend on the relative load of each service. mechanisms as part of our future work.

References

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

[2] "Microsoft tv: Iptv edition," http://www.microsoft.com/tv/IPTVEdition.mspx.

[3] 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)*, 2011.

[4] V. Aggarwal, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and V. Vaishampayan, "Exploiting Virtualization for Delivering Cloud-based IPTV Services," in *Proc. of IEEE INFOCOM (mini-conference)*, Shanghai, April 2011.

[5] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[6] N. V. Thoai and H. Tuy, "Convergent algorithms for minimizing a concave function," in *Mathematics of operations Research*, vol. 5, 1980.

[7] R. Urgaonkar, U. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proceedings of IEEE IFIP NOMS*, March 2010.

[8] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[9] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," in *Proc. of ACM Multimedia*, San Francisco, CA, October 1994, pp. 15–23.

[10] A. J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, "Deadline Scheduling for Real-Time Systems EDF and Related Algorithms," 1998, the Springer International Series in Engineering and Computer Science.

[11] L. I. Sennott, *Stochastic Dynamic Programming and the Control of Queueing Systems*. Wiley-Interscience, 1998.

[12] D. P. Bertsekas, "Dynamic Programming and Optimal Control," in *Athena Scientific, Blemont, Massachusetts*, 2007.

[13] G. Ramamurthy and B. Sengupta, "Delay analysis of a packet voice multiplexer by the ΣDi/D/1 Queue," in *Proceedings of IEEE Transactions on Communications*, July 1991.

[14] H. Tuy, "Concave programming under linear constraints," *Soviet Math 5*, pp. 1437–1440, 1964.

[15] S. Sergeev, "Algorithms to solve some problems of concave programming with linear constraints," *Automation and Remote Control*, vol. 68, pp. 399–412, 2007, 10.1134/S0005117907030034. [Online]. Available: http://dx.doi.org/10.1134/S0005117907030034