International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 6 | ISSN : 2456-3307

Integration of Multi Server for Profit Efficiency in Cloud Computing

V. Praveen[1], V. Gobu[2], M. Kavitha[3], T. Suvaikin Punitha[4]
[1, 2&4]Assistant Professor, N.S.N College of Engineering and Technology, Karur, Coimbatore, Tamilnadu, India
[3]Research Analyst, IsClor Soft Solutions, Coimbatore, Tamilnadu, India

ABSTRACT

Virtualized cloud-based services can take advantage of statistical multiplexing across applications to yield significant cost savings to the operator. Achieving similar benefits with real-time services can be a challenge. It seeks to lower a provider's costs of real-time IPTV services through a virtualized IPTV architecture and through intelligent time-shifting of service delivery. The merits of the differences in the deadlines associated with Live TV versus Video-on-Demand (VoD) to effectively multiplex these services. A generalized framework is provided for computing the amount of resources needed to support several services, without missing the deadline for any service. An optimization formulation that uses a generic cost function is build. The multiple forms for the cost function (e.g., maximum, convex and concave functions) to reflect the different pricing options are implemented. The solution to this formula gives the number of servers needed at different time instants to support these services. A simple logic for time-shifting scheduled jobs in a simulator and study the reduction in server load using real traces from an operational IPTV network is implemented. End results explain the load is minimized by ~ 24%. There are interesting open problems in designing mechanisms that allow time-shifting of load in such environments.

Keywords: Cloud Assets, IPTV Assistance

I. INTRODUCTION

As IP-based video delivery becomes more famous, the demands placed upon the service provider's resources have rapidly increased. Service providers typically provision for the peak demands of each service across the subscriber population. Provisioning for peak demands leaves resources under-utilized at all other periods. This is particularly ensured 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 every request being served by a server using a unicast stream. When end users change channels while watching live TV, we need to provide extra functionality to so that the channel change takes effect instantly. For each channel change, the user has to join the multicast group associated with the channel and wait for some time enough data to be buffered before the video is displayed. 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 a speeded up rate using a unicast stream from the server. The series of constituting a playoff buffer is filled quickly, and this keeps switching latency small. When the play out buffer is filled up to the play out point, the set top box goes 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 made 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 vast number of servers grouped in a data center for serving individual channels, and are scaled up as the number of customers increases. However this demand is transient and typically only lasts several seconds, possibly up to a couple of minutes. As results, many 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 usage is very bursty with a large peak to average ratio, VoD has relative manner in steady load and imposes “not so stringent” delay bounds. Most 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 try to minimize the resource requirements for supporting the service by taking advantage of statistical multiplexing across the different services - in the sense; we try to satisfy the peak of the sum of the demands of the services, rather than the sum of the large demand of each service when they are handled independently. Virtualization provides us the ability to share the server resources across these services.

In this paper, Our 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 deliver a popular channel). Other VMs would be developed 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 the ICC bursts (channel change behavior can be predicted from historic logs as a result of live TV show timings. The channel changes usually occurring every half an hour (see Figure 1)). Figure 1 also shows the respective VoD load and the aggregate load for both services together. In anticipation of the ICC workload, we try to accelerate delivery of VoD content (for example, for a very small number of minutes of play out time) to the end users' STBs and shift the VoD demand away from the ICC burst interval (see Figure 2). Also this will be ensured that VoD users will not notice any impairment in their delivered quality of service (e.g. frozen frames etc.) as the play out can be from the local STB cache.

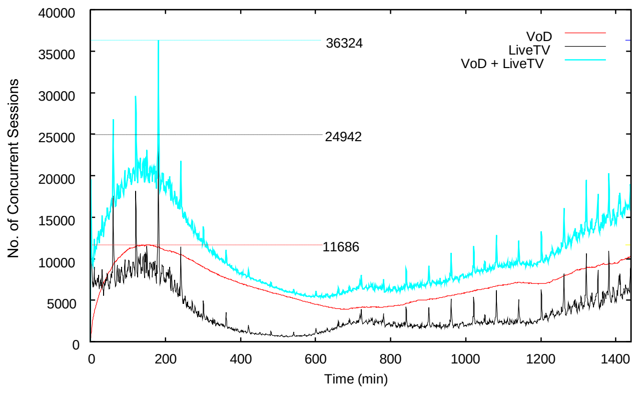

Figure 1. LiveTV ICC and VoD Concurrent Sessions (vs) Time, ICC Bursts Seen Every Half Hour

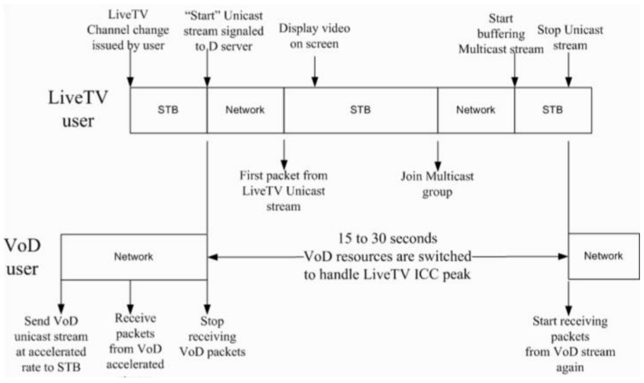
LiveTV Channel change issued by user
"Start" Unicast stream signaled to D server
Display video on screen
Start buffering Multicast stream
Stop Unicast stream
LiveTV user
STB
Network
STB
Network
STB
First packet from LiveTV Unicast stream
Join Multicast group
VoD user
Network
15 to 30 seconds
VoD resources 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

Figure 2. Live TV ICC and VoD Packet Buffering Timeline

In preliminary work on this topic [4], we analyzed the maximum number of servers that are needed to service jobs with a strict deadline constraint. We also assume non-causal information (i.e., all deadlines are known a priority) of the jobs arriving at each instant. In this paper, we kept in mind a generalized cost function for the servers. The cost of the servers in this model can be a function of time, load, etc. Our aim 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.

The sever-capacity region formed by servers at each time instant such that all the jobs arriving meet their deadlines is identified, which is explained as: the region such that for any server tuple with integer entries inside this region, all the deadlines can be met and for any server tuple with integer entries outside this region, there will be minimum one request that misses the deadline. Also we 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 the previous results in the literature where the numbers of servers are fixed at all times [5]. The server-capacity region is formed by linear constraints, and thus region is a polytypic.

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 and a separable convex function, or a maximum function. Note that even though the functions are concave/convex, the possibility set of server tuples is all integer tuples in the server-capacity region. This integer constraint leads the problem hard, in general. Show that for a piecewise linear separable convex function, an optimal strategy that redues the cost function can be easily described. This strategy only needs causal information of the jobs arriving at each time-instant. For any concave cost function, 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, best known concave programming techniques without integer constraints can be used to solve the problem [6]. Finally, for a maximum cost function, seek to minimize the maximum number of servers used over the entire period. This paper explains 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 namely, the maximum and piecewise linear convex cost functions. Set up a series of numerical simulations to see the effect of varying firstly and the ICC durations and secondly VoD delay tolerance on the total number of servers needed to accommodate the combined workload. 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, shows by means of a faithful simulator implementing both these services in paper, that a careful choice of a look ahead smoothing window can help to average the additional VoD load. Our approach only requires a server complex that is sized to meet the requirements of the ICC load, which is no deadline flexibility, and paper explains can almost completely mask the need for any additional servers for dealing with the VoD load.

II. NUMERICAL RESULTS

Maximum Cost Function

A series of experiments is setup to see the effect of varying firstly, ICC durations and secondly, VoD delay tolerance on the total number of servers needed to accommodate the added 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 rule described in computes the minimum number of concurrent sessions that need to be accommodated for the added workload. The legends in each plot indicate the duration that each VoD session can be delayed by the superposition of the number of VoD sessions with a periodic synthetic LiveTV ICC session.

The duration of the ICC session is set to be 15 seconds (i.e. all ICC activity come in a burst and lasts for 15 seconds), the peak of the pulse is set to be the peak of the VoD concurrent sessions for the entire 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 Five 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.

If the VoD service can afford only 1 second delay and then the total number of sessions that need to be accommodated is roughly double. A similar effect, the only difference here is that an operational trace is used for LiveTV. 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.

A. Convex Piecewise Linear Cost Function

Note how for different values of K allocates substantially different number of servers. 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 minutes 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. 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 K which is at-least given all the requests can be served at the lower cost and hence never are more than K servers needed, 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. Reduce the value of K; high jobs need to be served with higher cost. Portrays a similar story for an operational LiveTV trace. With a smaller K *value*, jobs are delayed to create larger overshoots (red spike of value 3.6×10^4).

III. SIMULATION

To demonstrate the efficacy of our proposal in the realistic situation of supporting IPTV services, implemented the adjustment mechanism in a custom event-driven simulator. Customers request traces were collected over 24 hours at a set of VHOs in one state for both VoD and ICC. Our data more than 18 million VoD and ICC requests in that time period. Result shows that a substantial reduction (~ 24% for our traces) in peak server load by adjusting the deadlines of VoD requests in anticipation of ICC requests.

A. Experiment Setup

The simulator models the customers and the server and represents by a simple small link, the complex network that typically connects them. We view each video as comprising of chunks. This is similar to what most commercial streaming systems (e.g., HTTP Live Streaming, Smooth Streaming) employ today. When a customer's requests a video, it is send the requests to the server and in response identifies the chunks in video. Set each chunk to be up to 30 seconds long. Each customer 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. Receiving a chunk, the client requests the next outstanding chunk.

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. Call this the *ICC Burst Window*. To minimize the load due to these ICC bursts by advancing the transfer of previously scheduled VoD requests. The process can adopt is depicted in Figure. 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 can call the *smoothing window*, advance the jobs already scheduled in the ICC burst window.

These jobs are served prior to their deadline. However, in this system can make room for the increased number of ICC requests expected in the burst window. As results, these requests will result in some continued VoD load during the burst window. One could however implement a most sophisticated scheme when the environment warrants it.

The reduction in load depends on multiple factors. First, need to predict when the burst will occur. Next, 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 start the adjustment process, its time period for averaging the load (i.e., smoothing window) also plays a key role.

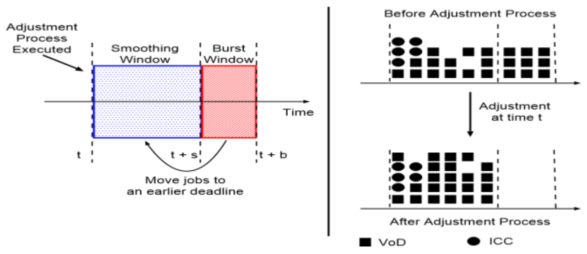

Figure 3. Overview of the Job Adjustment Process

Study each of these effects in our experiments. Assume that our ICC spike occur at fixed 30 minute intervals. By default assume that our burst window is one minute long, but also experiment with two minute bursts. 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 ten minutes. Note that ideally want to use a larger smoothing window than the burst window to spread out the load imposed by the moved jobs; otherwise will experience a spike in the load during the smoothing window, thus negating the benefit of a procedure somewhat. Primary metric is the number of concurrent streams that servers have to serve. Use this as our metric because it directly translates to how many servers are required to support the total request load. 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, shows separately plot the load due to VoD requests only, ICC requests, and the combined load for both VoD and ICC requests. Figure 1 explains that if we did not do any adjustment, would need to support a maximum of 36324 concurrent streams of VoD and ICC requests. If only supported ICC requests, this goes down to 24942 streams. This reduces further to 11686 streams considering VoD alone. Recall the ICC requests result in 15 seconds of data transfer and are served immediately (the deadline is 0). Hence the best can do is to go down to 24942 streams if we support both services 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.

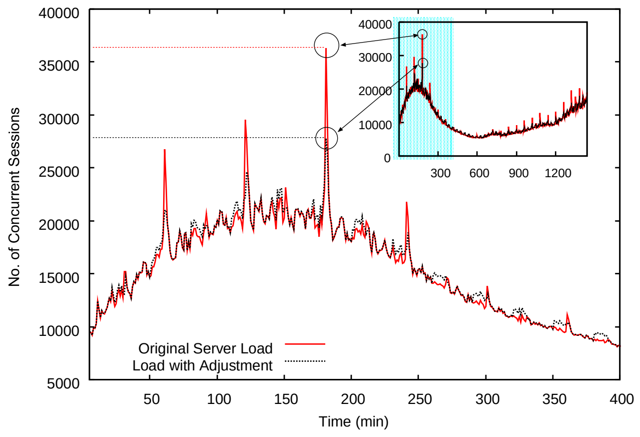

Figure 4. Reduction in Load Due to Job Rescheduling

C. Rescheduling Jobs Reduces Load

For the main results, shows that rescheduling jobs to an earlier deadline is indeed possible and that it can result in a significant reduction in the aggregate load. Assume an ICC burst window of one minute and a smoothing window of ten minutes. Also assume that all the VoD jobs prior to the burst window are moved to the beginning of the smoothing window.

The inset in Figure 10 explains the trends for the whole day. This paper plot peak period for the day (the first Four hundred minutes, marked by the shaded region in the inset) during which the peak number of streams are served. The result of this experiment presents that with the adjustment are able to bring the peak number of concurrent streams down from 36324 streams to 27813 streams, a ~ 24% reduction. This number is close to the load due to ICC requests alone, indicating that have successfully moved all the VoD requests are needed to 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. This Paper attribute the lower gain to the way exercise the time-shifting of the VoD load. Recall the adjustment of serving VoD requests is done at the start of the smoothing window. Any VoD requests that arrive after the adjustments are initiated cannot be rescheduled and results in load during the burst window as well. With a 10 minute smoothing window, see quite a few of the new VoD requests after the adjustment is complete. To understand this interaction better, 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. Selecting a small smoothing window results in more accurate determination of how many scheduled VoD jobs exist, but can result in a load spike within the smoothing window. On the other way a large smoothing window allows us the average the VoD load from the burst window better, but prevents the re-scheduling of many new VoD sessions that arrive subsequently. In this paper quantify the effect of the smoothing window, while keeping the burst window at two minutes. VoD jobs are shifted from those two minutes.

Interestingly, see that at the peak (around the 180 minute time marker), using a Five minute smoothing window results in better performance than a Ten minute smoothing window (26979 vs.

27813 streams). In this paper attribute the improvement to the ability to reschedule more VoD streams because of the smaller smoothing window. It is not as simple as just employing a smaller window. When reduce the smoothing window further, to two minutes, the load is consistently higher than the other windows. Even more importantly, the ten minute smoothing window consistently outperforms the others outside the peak viewing period. This is because at the peak period, the number of ICC requests are significantly higher than the VoD requests. This moving as many VoD requests as possible is important. At other ways, the number of VoD requests is higher. This moving all the VoD requests to an earlier deadline increases the load at that time of event. This is significant; it tells us that need a more sophisticated approach to predicting the load in a burst and in selecting 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 counterproductive. 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 moved 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 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, results are presented from a simple adjustment mechanism. 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.

IV. CONCLUSION

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 infrastructures. 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 accommodate 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.

V. 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," from the website 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