

Thwarting Virtual Bottlenecks in Multi-Bitrate Streaming Servers

Bin Liu and Raju Rangaswami
Florida International University
{bliu001,raju}@cs.fiu.edu

Zoran Dimitrijević
Google, Inc.*
zorand@gmail.com

Abstract

Current cycle-based disk IO schedulers for multi-bitrate streaming servers are unable to avoid the formation of virtual bottlenecks. We term a bottleneck as virtual when it occurs within a single resource subsystem, and it is possible to use a secondary under-utilized resource to thwart the bottleneck. We present stream combination, an IO scheduling technique that addresses this problem. Stream combination predicts the formation of virtual bottlenecks and proactively alters the IO schedule to avoid them. A simulation study suggests significant performance gains compared to the current state-of-the-art fixed time-cycle IO scheduler.

1 Introduction

The design goals of *guaranteed-rate IO* and *high throughput* within a streaming server requires establishing a trade-off between memory-use and disk-bandwidth utilization; this has been long recognized by designers of streaming multimedia systems [7, 8]. The underlying mechanism that determines this trade-off is the disk IO scheduling algorithm. Prior approaches to scheduling in real-time systems can be classified into two basic categories: *deadline-based priority scheduling* [3, 5, 8, 9] and *time-cycle-based scheduling* [1, 6, 7]. Deadline-based priority scheduling works excellently for CPU scheduling with provable guarantees for task completion. However, guaranteeing IO rate and performing admission control under this paradigm requires constant-overhead resource preemption [4], not feasible for disk-based systems.

The time-cycle-based IO scheduling model, originally proposed as quality proportional multi-subscriber servicing (QPMS) by Rangan et al. [7], is a simpler and more popular model for streaming media servers. This is due to the fact that it supports guaranteed-rate IO and a provably correct admission control mechanism [1]. In this model, each stream is serviced exactly one IO per time-cycle; the retrieved data is stored in a display buffer. The size of each IO is such that the display buffer does not underflow before the next IO for that stream.

In a multi-bitrate streaming server, the buffer sizes for different streams could vary significantly, implying that the corresponding IO sizes could also be vastly different. Intuitively, the disk utilization depends on the average IO size, since this metric directly dictates the overhead component. Lesser the average IO size, greater the fraction of per-unit time spent on access overheads, and lower the disk utilization. In the time-cycle model, the disk utilization therefore depends on the bi-

trate of the streams serviced in each time-cycle. If the average bitrate of streams serviced in a time-cycle is low, the average IO size and the achieved disk throughput are low, potentially resulting in a *virtual disk-bandwidth bottleneck*. We call this a virtual bottleneck because this bottleneck is a result of a mis-configured time-cycle and may be avoided. One way to avoid this bottleneck and increase disk throughput would be to increase the duration of the time-cycle. However, increasing the time-cycle suddenly would result in display buffer underflow. Second, the server memory requirement would also increase as a result, increasing faster than the achieved disk utilization. Chang et al. analyze memory requirements in streaming servers extensively in [1]. A solution which can increase the average request size, without severely impacting the memory use would eliminate this virtual disk-bandwidth bottleneck.

Virtual memory-bottlenecks can occur as a result of a high average bitrate of streams. Higher the average bitrate, larger are the display buffer sizes, and consequently, greater the total memory requirement. In such situations, time-cycle duration reduction can be used to potentially avoid this virtual bottleneck. However, this reduction cannot occur after the bottleneck has been established. Proactive and dynamic reduction of time-cycle duration has not been studied before.

In this paper, we propose *stream combination*, a variant of the time-cycle-based scheduling algorithm that dynamically adapts to changing system bottlenecks brought upon by shifting workloads. Stream combination provides guaranteed-rate IO and a provably correct admission control. Using a technique of combining and splitting IO streams and a technique for dynamic time-cycle alteration, it accounts for and avoids virtual disk- and memory- subsystem bottlenecks until these system resources are fully utilized.

2 Stream Combination

In this section, we present the rationale behind stream combination and the algorithm that drives this technique.

2.1 Rationale

Virtual bottlenecks can occur when servicing a dynamic streaming workload in either the memory or disk subsystem. Earlier, we noted that for virtual disk-bandwidth bottlenecks, simply increasing the time-cycle duration is not an acceptable solution. We investigate further to determine the root cause of disk IO inefficiency. For a stream with bitrate R serviced in a time-cycle of duration T , the amount of data retrieved in each IO is $R \times T$ and the amount of time spent to perform this IO is the sum of an (overhead) access time, T_{access} , and a data retrieval time, $\frac{R \times T}{R_{disk}}$, where R_{disk} is the data transfer rate from

*This work was performed when the author was at UC, Santa Barbara.

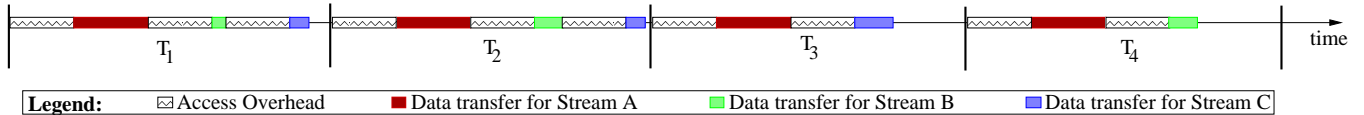


Figure 1. The Stream Combination Technique.

the disk medium. Therefore, the efficiency of the IO for the stream is:

$$e = \frac{R \times T}{R_{disk} \times T_{access} + R \times T} \quad (1)$$

Based on Equation 1, we note that a stream with high bitrate may have fair efficiency while a stream with low bitrate has poor efficiency. This raises the question: *Can we combine two or more low bitrate streams to obtain a single higher bitrate stream and improve IO efficiency?*

Figure 1 presents one possible combination technique. T_i denote time-cycle durations along a time axis. Streams A, B, and C are currently being serviced by the system. The bitrate of A is relatively high compared to B and C. In time-cycle T_1 (prior to combination), the IO scheduler performs one IO each per time-cycle per stream, retrieving S_A , S_B and S_C amount of data respectively. The scheduler starts the stream combination process in time-cycle T_2 by retrieving twice the amount of data for stream B ($= 2 \times S_B$). In time-cycle T_3 , the scheduler does not perform IO for Stream B, but retrieves twice the amount of data for stream C ($= 2 \times S_C$). Starting from time cycle T_3 , in any given IO cycle, only one of streams B or C are serviced, reducing the number of access overheads by one, increasing the average IO size, and consequently improving disk utilization. Although it is possible (and indeed practical) to combine more than two streams at a time as well as further combining previously combined streams, we do not explore this direction in this paper and leave it to future work.

Although such a technique improves disk utilization as a result, several issues must be considered in a combination strategy: (i) the state of the system; combination makes sense only if disk-bandwidth is the bottleneck, (ii) combination must be proactive and must not allow the system to reach a bottleneck state before taking effect, (iii) how many streams must be combined to avoid the virtual bottleneck? (iv) combination increases memory requirement, and a wrong combination decision may potentially result in a virtual memory-bottleneck, (v) the combination operation incurs a transitory data transfer overhead during the time-cycle in which combination is initiated, and (vi) after combination, if there is a virtual memory-bottleneck at some later time due to shift in the workload, is *un-combining* or *splitting* combined streams straightforward?

The second virtual bottleneck is memory consumption. Assume that the first K out of N streams served by the system are in the combined state. If R_i is the bitrate of stream i and T denotes the time-cycle duration, the total memory requirement for N streams is the sum of the display buffer sizes of all

streams and is given by:

$$M = \sum_{i=1}^N T \times R_i + \sum_{i=1}^K T \times R_i \quad (2)$$

This equation follows from the observation that combined streams require buffering for two time-cycle durations as opposed to one time-cycle duration for uncombined streams. When the system approaches a potential virtual memory-bottleneck, it may be in one of two states: (a) there exist combined streams in the system, and (b) all streams are uncombined. In case (a), combined streams can be split to reclaim memory. In case (b), reducing time-cycle duration can reduce total memory requirement. However, three issues must be considered: (i) if several combined streams exist, which stream must be chosen to split first? (ii) how many combined streams should be split to avoid the bottleneck? (iii) by how much must the duration of the time-cycle be reduced to avoid the bottleneck? The answer to the question of which combined streams should be split first is straightforward. Splitting should be performed first on the high bitrate streams because they allow reclaiming the maximum amount of memory. However, the other issues need further investigation.

2.2 Mechanism

The basic idea of stream combination is to thwart virtual bottlenecks in streaming servers by proactively balancing memory and disk resource consumption under shifting stream workload. This balancing act is performed until both memory and disk resources are fully utilized. To balance these resources, we use two parameters, the memory utilization (u_m) and the time-cycle utilization (u_t). Memory utilization is the ratio of the utilized memory to the available memory, while time-cycle utilization is the ratio of the utilized time-cycle to the time-cycle duration. These parameters capture the relative availability of memory and disk-bandwidth resources.

A simplistic stream combination mechanism requires keeping track of u_m and u_t ; when $u_m < u_t$, it combines the two lowest bitrate uncombined streams; when $u_m > u_t$, it un-combines or splits the highest bitrate combined stream. However, this straightforward strategy has several problems: (i) when choosing to combine, there may be no uncombined streams, (ii) when choosing to split, there may be no combined streams, (iii) this simplistic strategy would typically result in frequent combinations and splits, and (iv) several combination operations in a short duration can lead to a significant transitory disk-bandwidth overhead for transferring additional data for combined streams.

To avoid these problems, the stream combination IO scheduler uses four heuristics:

```

Input:      Current Workload (W),
           Current Schedule (CS)
Output:    New Schedule (NS)

Procedure: CheckSchedule {
  Compute {u_m,u_t} from {W,CS} ;
  If(((u_m>u_mT || u_t>u_tT) && abs(u_m-u_t)<u_dT)
      || SFLAG) { Call Reschedule ; }
}

Procedure: Reschedule {
  SFLAG = false ;
  If(u_m>u_t) {
    If(combinedStreamsExist) {
      Split highest bitrate combined streams ;
      Modify schedule to NS ;
    } Else { Decrease Time-cycle by UNIT ; }
    Recalculate {u_m,u_t} from {W,NS} ;
    If(abs(u_m-u_t)>u_dT) { SFLAG = true ; }
    return NS ;
  } Else {
    If(uncombinedStreamsExist) {
      Combine lowest bitrate uncombined streams ;
      Modify schedule to NS ;
    } Else { Double Time-cycle duration ; }
    Recalculate {u_m,u_t} from {W,NS} ;
    If(abs(u_m-u_t)>u_dT) { SFLAG = true ; }
    return NS ;
  }
}

```

Figure 2. Stream Combination Scheduler.

1. When combination is required and no uncombined streams exist, the scheduler doubles the duration of the time-cycle, effectively un-combining all streams. Notice that this increase in time-cycle duration incurs no overhead.
2. When splitting is required and no combined streams exist, the scheduler decreases the time-cycle by a UNIT percentage value, thereby reducing memory requirement. However, the disk utilization degrades due to a reduced average IO size. Here, we trade disk-bandwidth to conserve memory.
3. It makes provision for three constants, the memory utilization threshold (u_mT), the time-cycle utilization threshold (u_tT), and the difference threshold (u_dT). The decision to reschedule is made only in case either memory or time-cycle utilizations exceed their threshold and their difference is greater than the difference threshold.
4. When a decision to combine or split is made, the scheduler spreads out multiple required combine or split operations, allowing only one operation per time-cycle, thereby minimizing the transitory disk-bandwidth overhead. This is achieved using a scheduling flag (SFLAG).

The detailed IO scheduling algorithm is presented in Figure 2. The procedure CheckSchedule is invoked at the beginning of each time-cycle, which in turn invokes the Reschedule procedure if required.

3 Experimental Evaluation

To evaluate the performance of the stream combination IO scheduler, we built a simulator to compare it with a fixed time-cycle scheduler. The system was configured to have 128MB

of total available memory to buffer stream data. The maximum disk transfer-rate was 50MB/s and the average disk access time (including seek, rotational, and settle overheads) was 10ms. The base-line IO scheduler chosen was Fixed-Stretch [1], a state-of-the-art fixed time-cycle scheduler that balances disk-bandwidth and memory use.

Figure 3 tracks the following metrics during a simulation run of 20 minutes for a workload with uniformly distributed stream bitrates between 128 and 1024 kbps and with uniformly distributed request inter-arrival times between 2-7 seconds: (a) memory consumption (in MB), and (b) number of streams in service at any instant. The initial time-cycle duration for the stream combination scheduler was the same as that of the fixed time-cycle scheduler: 500 milliseconds. Initially, as streams arrive, the two scheduling strategies performed similarly. At approximately 200 seconds, the fixed time-cycle scheduler encountered a virtual disk-bandwidth bottleneck due to an underestimated time-cycle duration. The stream combination scheduler detected the future formation of a virtual disk-bandwidth bottleneck and proactively started combining streams at approximately 100 seconds. As a result, it successfully thwarted the bottleneck. Beyond 200 seconds, the fixed time-cycle scheduler was unable to accommodate more number of streams in a time-cycle. Our scheduler was able to continue servicing greater number of streams in each time-cycle, delivering as much as 55% more throughput than the fixed time-cycle scheduler.

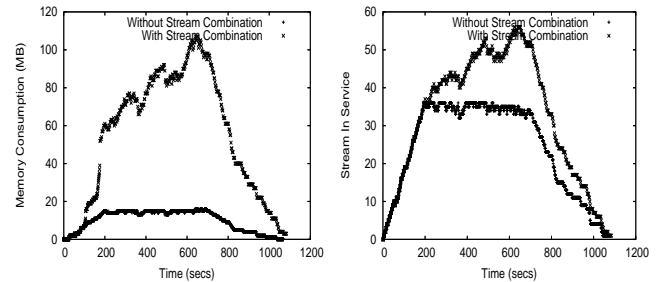


Figure 3. Comparison for a time-cycle=500ms.

Figure 4 demonstrates the case where the initial time-cycle duration for both schedulers was set to 5 seconds. The generated workload was the same as for the previous experiment. At around 200 seconds into the simulation, the fixed time-cycle scheduler encountered a virtual memory-bottleneck that limited its throughput. Our scheduler proactively started reducing the duration of the time-cycle (and the memory consumption as a result) at around 100 seconds (See Figure 4(b)) to successfully thwart the virtual bottleneck. More time-cycle reductions occurred beyond 200 seconds, dynamically adapting to the increased workload and delivering as much as 100% more throughput than the fixed time-cycle scheduler.

The above experiments demonstrated the inadequacy of a statically chosen time-cycle duration. We now determine how the throughput of the streaming server (in terms of the maximum number of streams admitted) depends on the time-cycle duration. Figure 5(a) compares against a fixed time-cycle

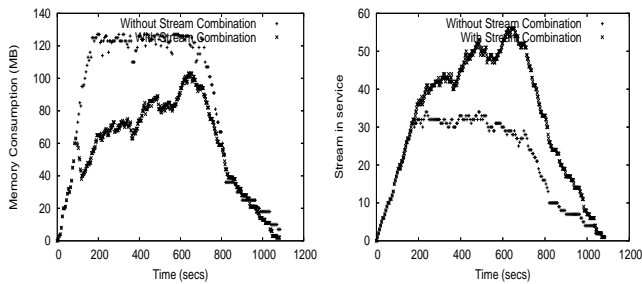
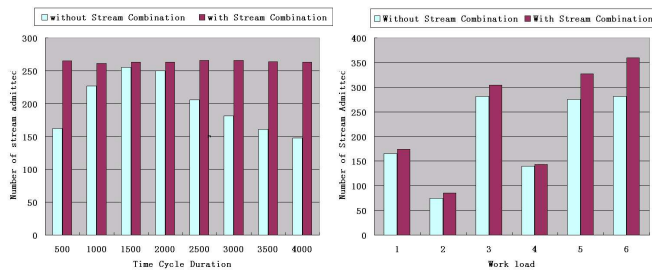


Figure 4. Comparison for a time-cycle=5000ms.

scheduler with different time-cycle durations in each experiment, shown along the X-axis. The workload used was the same as for the previous experiments. As the initial time-cycle duration is changed, the fixed-time cycle scheduler admitted different number of streams, achieving its maximum for a time-cycle duration of 1.5 seconds. With stream combination, regardless of the initial time-cycle duration, the scheduler dynamically altered both its schedule as well as time-cycle duration to always provide the maximum throughput. It is important to note that, in case of the fixed time-cycle scheduler, determining the optimal time-cycle duration requires prior knowledge of the workload. Second, for real-world streaming servers, the natural shift in the workload over time precludes the existence of an optimal time-cycle duration. In such real-world scenarios, the stream combination scheduler dynamically adapts to deliver the maximum possible throughput.



(a) Varying time-cycle duration.

(b) Varying workload.

Figure 5. Throughput comparison.

Our final experimental result, depicted in Figure 5(b), compares the relative performance for six different workloads. These workloads were generated by varying both the distribution of stream bitrates as well as the arrival rates. Workloads #1-3 used stream bitrates generated from a uniform distribution. The time-cycle scheduler picked the time-cycle duration based on the average duration (assuming prior knowledge) and performed within 8% of the stream combination scheduler. Workloads #4-5 used a non-uniform distribution for stream bitrates; #4 favored high bitrates and #5 favored low bitrates. With workload #4, the primary bottleneck is memory and for #5, it is disk-bandwidth, with no virtual bottlenecks formed during these simulations. Even so, the stream combination scheduler was able to fine-tune the time-cycle duration to deliver as much as 15% more throughput for workload #5. Finally workload #6 varied the distribution over time to initially

favor low bitrates and then high bitrates. The fixed time-cycle scheduler did not have a clear choice for the time-cycle duration and used the average bitrate as the basis. The stream combination scheduler dynamically varied the time-cycle duration over time to better match the request traffic and delivered as much as 30% more throughput. It is important to note that real-world streaming workloads behave relatively more like workload #6 (probably with greater variations) than like workloads #1-5, underscoring the importance of stream combination.

4 Conclusions and Future Work

We have presented stream combination, an IO scheduling technique that avoids virtual bottlenecks in streaming servers. This technique predicts subsystem bottlenecks and proactively alters the IO schedule to successfully thwart them until all system resources are fully utilized. Stream combination achieves its goal using the dynamic techniques of combining low-bitrate streams, splitting high-bitrate combined streams, and changing the time-cycle duration, as required. A simulation study suggests that this technique can offer significant performance improvement over fixed time-cycle schedulers. An implementation of the stream combination technique is currently being incorporated into Xstream [2], a real-time streaming multimedia system. In the future, we plan to evaluate the appropriateness of the family of deadline-based priority schedulers for real-time disk IO scheduling and compare it against the stream combination scheduler.

References

- [1] E. Chang and H. Garcia-Molina. Effective Memory Use in a Media Server. *Proceedings of the 23rd VLDB Conference*, pages 496–505, August 1997.
- [2] Z. Dimitrijevic, R. Rangaswami, and E. Chang. The Xstream Multimedia System. *Proceedings of the IEEE Conference on Multimedia and Expo*, August 2002.
- [3] K. Jeffay, D. F. Stanat, and C. U. Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. *Proceedings of the Twelfth IEEE Real-Time Systems Symposium*, December 1991.
- [4] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *ACM Journal*, January 1973.
- [5] A. Molano, K. Juvva, and R. Rajkumar. Guaranteeing Timing Constraints for Disk Accesses in RT-Mach. *Proceedings of the IEEE Real Time Systems Symposium*, 1997.
- [6] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A Low-cost Storage Server for Movie On Demand Databases. *Proc. VLDB*, September 1994.
- [7] P. V. Rangan, H. M. Vin, and S. Ramanathan. Designing and On-Demand Multimedia Service. *IEEE Communications Magazine*, 30(7):56–65, July 1992.
- [8] A. L. Reddy and J. Wyllie. Disk Scheduling in a Multimedia I/O System. *Proceedings of the ACM Conference on Multimedia*, pages 225–233, 1993.
- [9] J. C. Wu and S. A. Brandt. Storage Access Support for Soft Real-Time Applications. *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 164–173, May 2004.