On-line Stream Merging in a General Setting*

Wun-Tat Chan Tak-Wah Lam†
Hing-Fung Ting‡ Prudence W.H. Wong‡
Department of Computer Science, University of Hong Kong, Hong Kong
{wtchan,twlam,hfting,whwong}@cs.hku.hk

May 8, 2002

Abstract

This paper is concerned with on-line scheduling algorithms for merging streams in a video-on-demand (VOD) system so as to minimize the server bandwidth. We present the first algorithm that has a constant competitive factor (precisely, 5). Our algorithm, unlike previous ones, is not limited to the scenario where clients are equipped with large buffer and client receiving bandwidth. It remains 5-competitive in all settings of buffer size and receiving bandwidth. Technically speaking, our algorithm is based on a novel observation that the behavior of any schedule can be modeled by a rectilinear (binary) tree on a grid. This observation eases the analysis of our algorithm as well as the optimal algorithm.

1 Introduction

This paper studies an on-line scheduling problem arising from video-on-demand (VOD) systems. A VOD system often receives many requests for a hot video over a short period of time (say, Friday 7 PM to 9 PM). If the video server uses a dedicated video stream to serve each request, the bandwidth requirement for the server becomes enormous. A simple alternative is to batch [1, 8, 10] the requests so that a single stream is issued to serve requests arriving over a period of time; however, this increases the response time substantially.

Stream merging is a commonly used approach to reduce the bandwidth requirement without sacrificing the response time. There are two models to enable stream merging: *skimming* is based on the hardware architecture that each client is equipped with extra buffer and can receive data from two streams simultaneously [7, 13, 15, 18], and *piggybacking* is based on the alteration of the play rate of the streams [2, 14, 16]. The deviation of

*This paper combines results of two conference papers: (1) A 5-Competitive On-line Scheduler for Merging Video Streams. In Proceedings of the 15th International Parallel and Distributed Processing Symposium, 2001, pp. 2165–2172. (2) Improved On-line Stream Merging: from a Restricted to a General Setting. In Proceedings of the 7th International Computing and Combinatorics Conference, 2001, pp. 432–442.

†This research was supported in part by Hong Kong RGC Grant HKU-7024/01E.

‡This research was supported in part by Hong Kong RGC Grant HKU-7103/99E.

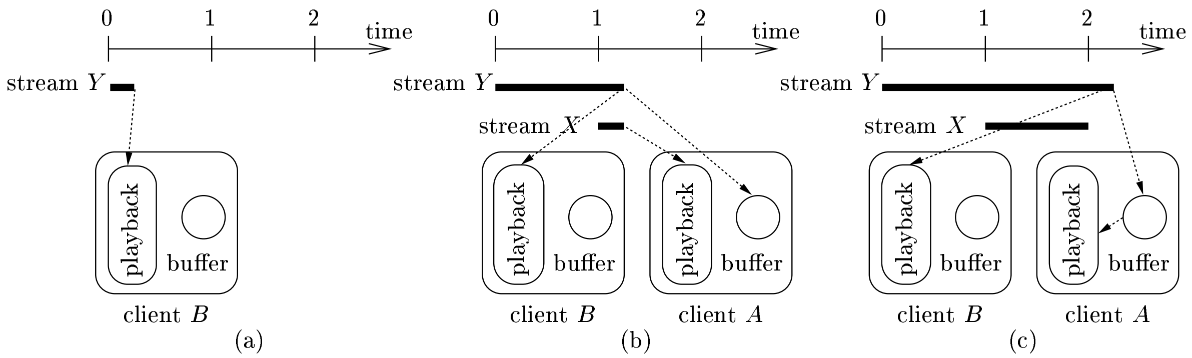


Figure 1: (a) At time 0, client B is served by a new stream Y. (b) At time 1, client A arrives. A is served by another new stream X, and it simultaneously buffers data from Y. (c) At time 2, X terminates. A plays back using the content of its buffer and continues further buffering from Y. Assume the video comprises 120 time units. The total bandwidth for serving both clients is reduced from 240 to 121.

the play rate cannot be too large and this limits the amount of bandwidth that can be saved by the piggybacking model (see e.g. [7]). In this paper the stream merging model in concern is skimming. In its simplest setting, stream merging works as follows. A client initially receives data from a dedicated stream X for immediate playback. At a certain time, the client may start to buffer data simultaneously from another stream Y initiated Δ time units earlier than X (due to another client). Note that Y is Δ units ahead of X. After listening to both X and Y for Δ time units, the client no longer needs X as it can play back the next Δ time units using the data in its buffer, while further buffering data from Y simultaneously. Hence, X may terminate now and clients of X all switch to Y. In this case, we say that X *merges* with Y. Note that Y itself may later merge with another earlier stream. The stream merging technique allows more than one client to be eventually served using only one full stream, thus saving the bandwidth of the server. Figure 1 shows an example illustrating how bandwidth is saved.

Note that the above example assumes that a client has an extra buffer to store up to Δ units of video and can receive data simultaneously from two streams. The latter means that the client has a receiving bandwidth at least twice the minimum bandwidth required for playback only. As observed by many researchers (e.g., [4, 12, 13]), clients in reality may have only limited extra bandwidth and buffer. That is, a stream merging system is in general characterized by two positive integer parameters (α, β) with the meaning that every client has extra bandwidth to receive a unit of video in α units of time and a buffer storing up to β units of video. Note that a bigger α means smaller extra bandwidth. For instance, the example in Figure 1 assumes that $\alpha = 1$.

The stream merging problem is to find an on-line algorithm to determine for each existing stream whether to continue to run or merge with an earlier stream. Such an algorithm operates without the knowledge about future requests and the goal is to minimize the total bandwidth used by the server, i.e., the total duration of all streams. We analyze the worst-case performance of an on-line algorithm using competitive analysis [5]. An on-line algorithm is said to be c-competitive if, for any sequence I of inputs, it uses at most c times the bandwidth used by the optimal off-line scheduling algorithm for serving I. In this case, the competitive ratio of the algorithm is said to be at most c.

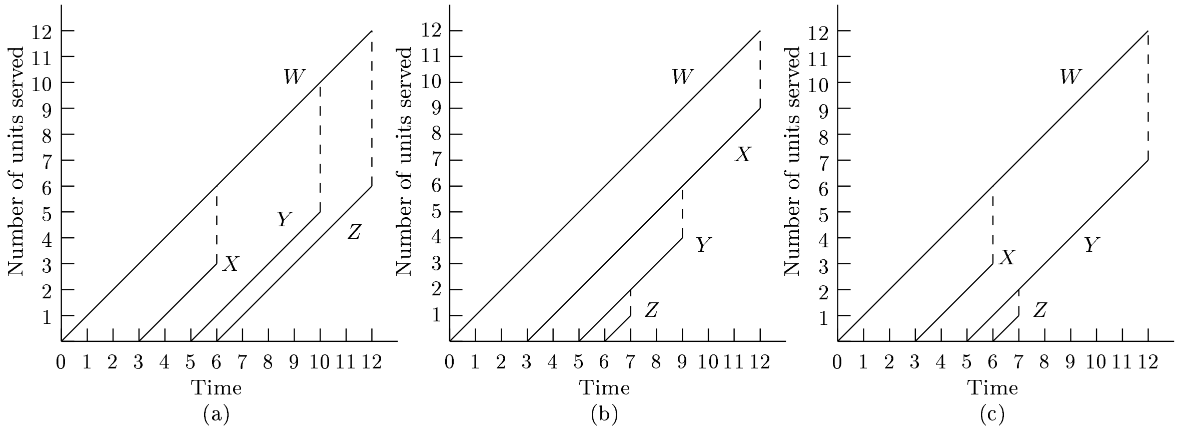


Figure 2: Three schedules for merging four streams with a total bandwidth of 26, 26 and 23, respectively. They illustrate that sometimes it is more beneficial for a stream to delay merging. (a) X, Y and Z merge with W in turn. The total bandwidth is 26. (b) X delays the merging with W; Y delays the merging with X; and Z merges with Y, X and then W. The total bandwidth is 26. (c) X merges with W; Y delays the merging with W; and Z merges with Y, which then merges with W. The total bandwidth is 23.

At first glance, the stream merging problem might look simple. One might expect that every stream should simply merge with an earlier stream as soon as possible. Figure 2 shows an example illustrating that in some case it may be more beneficial for a stream to delay merging; also, merging with the nearest stream may not yield the optimal schedule.

Previous results: Stream merging was first studied in a restricted form known as patching [6, 15, 17] or tapping [7], in which a stream is allowed to merge with a full stream only (the latter will not further merge with other streams). Eager *et al.* [11, 13] are the first to study cascaded merging of streams. They focused on the typical setting with $\alpha = 1$ and $\beta = \ell/2$, where ℓ is the length of a full stream, and suggested on-line algorithms that perform well empirically. Recently, Bar-Noy and Ladner [4] initiated the study of worst-case performance, showing that simple algorithms like best fit and nearest fit are indeed $\Theta(n/\log n)$-competitive, where n is the number of requests. More importantly, they gave an on-line algorithm with competitive ratio at most $\min\{\log_\phi n, \log_\phi \frac{1}{2D}\} + O(1)$, where $\phi = (1 + \sqrt{5})/2$, and $D < 1$ is the guaranteed start-up delay measured as a fraction of the time for a full stream. Note that the Bar-Noy-Ladner algorithm is still based on the setting with $\alpha = 1$ and $\beta = \ell/2$. Actually these algorithms still work if $\beta \geq \ell/2$. But in general, when $\alpha > 1$ and/or $\beta < \ell/2$, only empirical results have been known [12]. Whether there exists a worst-case performance guarantee for a general setting (α, β) has been an open problem [4].

Our result: In this paper we present the first on-line algorithm for stream merging that has a constant competitive ratio (precisely, 5). In other words, the performance guarantee does not deteriorate even if there are many requests and the start-up delay is required to be very small. More interestingly, our work resolves the open problem for general setting in the affirmative as our algorithm remains 5-competitive for all combinations of buffer size and extra bandwidth. Our result is based on a novel observation that stream merging in an optimal schedule cannot happen at arbitrary time and the behavior of any scheduler can be modeled by rectilinear trees on a grid. Such an observation eases the analysis of our algorithm as well as the optimal algorithm.

The remainder of the paper is organized as follows. Section 2 gives the preliminaries and Section 3 defines the binary merge tree representation which models merging schedules. In Sections 4 and 5, we show how to handle some special request sequences. Then in Section 6, we generalize our result to arbitrary request sequences. Finally, we give some remarks in Section 7.

2 Preliminaries

We assume that time is divided into intervals of unit length; each unit can be arbitrarily small. Requests arriving within the same interval are treated as a single request as they can all be served by multicasting one stream. At any time when a request arrives, a new stream is initiated immediately to serve the request. A stream is said to be a *partial* stream if it terminates before the end; otherwise it is a *full* stream. Given a sequence of requests arriving at different times, a schedule specifies for every stream initiated whether the stream merges with an earlier stream, and if so, when and with which stream it will merge. In our discussion, we distinguish two costs of a schedule. The *full cost* is the total duration of all the streams in the schedule, i.e., the bandwidth used by all the streams, and the *partial cost* is the total duration of all the partial streams in the schedule. An *optimal* schedule is one with the minimum full cost. For any schedule S, we denote by $f(S)$ and $p(S)$ the full cost and partial cost of S, respectively.

Below we review some basic concepts of stream merging that have been discussed in the literature [4, 11, 13]. For a stream X to merge with another stream Y initiated Δ time units earlier, a necessary condition is that the client buffer size is at least Δ. Suppose after X merges with Y, Y further merges with another stream Z initiated Δ' time units before Y. This means that clients of X, after switching to Y, is required to further buffer Z and eventually listen solely to Z via a buffer of size $\Delta+\Delta'$. Thus, a necessary condition for Y to merge with Z is that the client buffer size is at least $\Delta+\Delta'$. Another concern of such cascaded merges is that clients of Y cannot start buffering Z until the merging of X with Y is completed; otherwise, clients of X, while listening to X, have to buffer more than one stream (i.e., Y and Z) simultaneously. In general, a sequence of cascaded merges can happen only if the client buffer size is at least the maximum initiation delay among the streams involved. Also, each merge must be completed before the next one starts.

Next, we turn to the general setting. Let α and β denote the client extra bandwidth and buffer size, respectively. Note that both parameters are measured in terms of time units; in particular, α is the number of time units required to receive an extra unit of video. Consider two streams X and Y such that Y is initiated Δ time units earlier than X. Suppose clients of X start to buffer data from Y at time t. Then X can merge with Y at time $t+\alpha\Delta$ (see [12] for details of the buffering scheme). At time $t+\alpha\Delta$, Y should have played at least $\Delta+\alpha\Delta$ but no more than ℓ units; thus, $\Delta \leq \ell/(1+\alpha)$. As mentioned earlier, we require that $\beta \geq \Delta$ no matter X merges with Y directly or through a sequence of merges. Thus, a necessary condition for X to merge with Y is that $\Delta \leq \min\{\beta, \ell/(1+\alpha)\}$. We say that a request sequence is *compact* if the last request arrives within $K = \min\{\beta, \ell/(1+\alpha)\}$ time units from the first request.[1] Note that among

[1] Compact request sequences are a generalization of a similar notion in the work of Bar-Noy and

the streams initiated for a compact request sequence, it is feasible for any stream to merge with another stream through a sequence of one or more merges.

In Sections 3–5, we derive a 3-competitive on-line algorithm for handling compact request sequences. Then in Section 6, we show that a greedy approach can extend this algorithm to handle general request sequences.

3 Representation of schedules

In this section we show, under the assumption that the request sequence is compact, some interesting properties of the optimal schedules. This allows us to confine to schedules that satisfy these properties and represent such schedules using a simple model called *binary merge trees*.

In Section 3.1 we give the definition of *binary merge trees*, and in Section 3.2 we consider binary merge trees that are "embeddable" on a special grid. Binary merge trees or embeddable binary merge trees cannot represent all possible schedules for a compact request sequence. For example, they cannot model two or more streams merge with another stream at the same time. Nevertheless, in Section 3.3 we show that every optimal schedule can be represented by a binary merge tree that is embedded on a special grid. In Sections 4 and 5 we will give an on-line algorithm which produces a schedule represented by such a binary merge tree and compare its performance with an optimal binary merge tree.

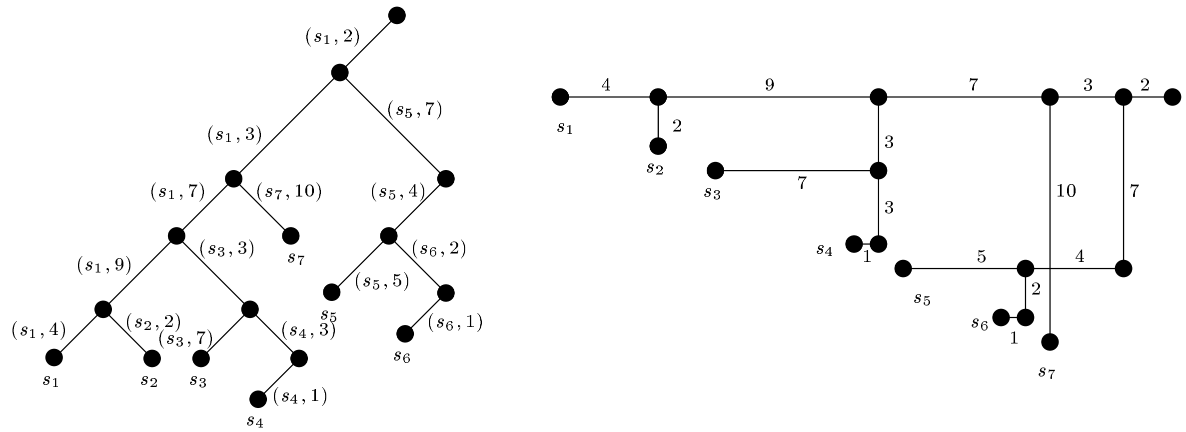
3.1 Binary merge trees

Binary merge trees attempt to model the occurrences of merging and the status of individual streams. This is in contrast to the merge trees used by Bar-Noy and Ladner [4], which model the relationship among the streams. On the other hand, binary merge trees are more restricted as they can only model schedules in which all streams except the first one merge with another stream eventually and every merging is binary in nature (i.e., no two or more streams merge with another stream at the same time).

Consider any compact request sequence I. A binary merge tree for I is a rooted, ordered binary tree satisfying the structural property that for any internal node u that has out-degree one, its only son must be the left son and u is the right son of its parent (or u is the root). The leaves have an one-one correspondence with the streams initiated for individual requests of I. Figure 3(a) gives an example.

In a binary merge tree, leaves model the status of individual streams and internal nodes model the occurrences of merging. Precisely, every edge e represents a stream s_e running for a duration of d_e time units. If e is adjacent to a leaf, then s_e is the stream associated with the leaf. Otherwise, we denote v as the endpoint of e that is farther away from the root, and s_e is the same as the stream associated with the left edge of v. The lifespan of a stream s is represented by a path starting from the corresponding leaf up to and including the first edge that is the right edge of some node or adjacent to the root. The total duration of s is the total durations of all the edges on this path. To ease our discussion, we define, for any node w on this path, $d(s, w)$ to be the sum of the durations over the edges from the leaf up to w. The leftmost path of the tree corresponds to the

Ladner [4], which considers the setting with $\alpha = 1$, $\beta = \ell/2$, and $K = \ell/2$.

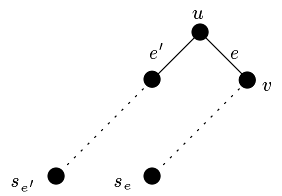


(a) A binary merge tree T. (b) The rectilinear drawing $\mathcal{D}(T)$.

Figure 3: Suppose $\alpha = 1$, $\beta = 11$ and $\ell = 25$. (a) A binary merge tree T involving seven streams. Note that the label of an edge e is (s_e, d_e). (b) The rectilinear drawing of T. Note that the length of an edge e equals d_e.

first stream, which does not merge with any other stream. The total duration of this stream is exactly ℓ.

An internal node u with two sons represents the occurrence of a merging as follows. Let v, e and e' be the right son, right edge, and left edge of u, respectively. Let Δ be the difference between the initiation time of stream s_e and stream $s_{e'}$. We require that $d_e = \alpha\Delta$. After s_e has played for $d(s_e, v)$ time units, clients of s_v start to buffer $s_{e'}$. Then, s_e merges with $s_{e'}$ at the time when s_e has played $d(s_e, v) + d_e$ time units.



The *full cost* of a binary merge tree is the sum of the durations over all edges, and the *partial cost* is the full cost minus the durations of the edges on the leftmost path. A binary merge tree for a given sequence is said to be *optimal* if its full cost is smallest among all possible trees. By construction, a binary merge tree defines a merging schedule with the same full cost and partial cost.

3.2 Rectilinear embedding

Binary merge trees (BMT) are better understood when they are embedded on a grid. Such an embedding facilitates the design and analysis of our algorithm. Details are as follows. Let T be any BMT, and let $\mathcal{D}(T)$ denote the following "rectilinear drawing" of T.

- Every left (resp. right) edge e of T is represented by a horizontal (resp. vertical) line with length equal to the duration associated with e.
- Every node u of T is represented by a point with its left edge and right edge laid respectively on its left and below.

See Figure 3(b) for an example. Notice that the first stream of T is represented by a contiguous horizontal line at the top. $\mathcal{D}(T)$ can be constructed recursively starting from the root. Due to the length requirement, edges in $\mathcal{D}(T)$ may cross each other. We say that $\mathcal{D}(T)$ is *planar* if no two edges (lines) share a point that does not represent a node of T.

Let G be a grid in which rows and columns are numbered $0, 1, 2, \cdots$ from top to bottom and left to right, respectively. G is "distorted" in such a way that every horizontal grid arc is of length $1 + \alpha$ and every vertical grid arc is of length α. The grid point on row r and column c is referred to as $[r, c]$. We generalize the notation $[r', c']$ to allow r' and c' being real numbers so as to represent a non-grid point. For any two points x, y on the same row, the notation $\delta_h(x, y)$ denotes the length of the horizontal grid line between x and y. For example, $\delta_h([10, 0], [10, 2.5]) = 2.5(1 + \alpha)$.

Suppose we superimpose $\mathcal{D}(T)$ on G such that the top-leftmost leaf (i.e., the leaf associated with the first stream) is put at the grid point $[0, 0]$. See Figure 4 for an example. By definition, the length of every vertical (right) edge in $\mathcal{D}(T)$ is a multiple of α; thus, every node of $\mathcal{D}(T)$ must lie on a horizontal grid line. The next lemma further shows that every leaf resides in a specific grid point. However, this may not be true for internal nodes.

Lemma 1. *With respect to the above way of superimposing $\mathcal{D}(T)$ on G, a leaf associated with a stream initiated t time units after the first stream is at grid point $[t, t]$.*

Proof. Let $u_1, u_2, \cdots, u_m$ be the leaves of T in the order of their initiation time. For each u_j, let s_j and t_j be the associated stream and the initiation time. Without loss of generality, we assume $t_1 = 0$. We prove Lemma 1 by an induction on these leaves. For the base case, the leaf u_1 is put at grid point $[0, 0]$ by construction.

Suppose the lemma is true for all leaves $u_1, \cdots, u_{i-1}$. We prove the case for u_i as follows. Along the path from u_i to the root, we denote w as the first node that is the right son of another node v. I.e., v represents the merging of s_i with another stream $s_{i'}$ for some $i' < i$, which is associated with leaf $u_{i'}$. By induction hypothesis, $u_{i'}$ is on row $t_{i'}$. By construction, $u_{i'}$ and v lie on the same row, and u_i and w lie on the same row. The edge (v, w) is represented by a vertical line of length $\alpha\,(t_i - t_{i'})$. Thus, w and u_i lie on row $(t_{i'} + (t_i - t_{i'})) = t_i$.

When s_i merges with $s_{i'}$, s_i has played $d(s_i, v) = d(s_i, w) + \alpha\,(t_i - t_{i'})$ time units, and $s_{i'}$ has played $d(s_{i'}, v)$ time units. Thus, $t_{i'} + d(s_{i'}, v) = t_i + d(s_i, w) + \alpha\,(t_i - t_{i'})$, or equivalently, $d(s_{i'}, v) - d(s_i, w) = (1 + \alpha)\,(t_i - t_{i'})$. Let γ be the point $[t_i, t_{i'}]$, which is on the same row as u_i. To show u_i lies on column t_i, we prove that $\delta_h(\gamma, u_i) = (1+\alpha)(t_i - t_{i'})$. By construction, $\delta_h(\gamma, u_i) + \delta_h(u_i, w) = \delta_h(u_{i'}, v)$. Thus, $\delta_h(\gamma, u_i) = \delta_h(u_{i'}, v) - \delta_h(u_i, w) = d(s_{i'}, v) - d(s_i, w) = (1 + \alpha)\,(t_i - t_{i'})$. In conclusion, u_i lies at the grid point $[t_i, t_i]$. $\square$

We say that a binary merge tree T is *G-embeddable*, or simply *embeddable*, if every node of $\mathcal{D}(T)$, except possibly the root, lies at a grid point when we superimpose T on G. Following the above discussion, we can have a simple condition for T to be G-embeddable.

Fact 2. *T is G-embeddable if and only if the length of every horizontal edge in $\mathcal{D}(T)$, or equivalently, the duration associated with every left edge in T, is a multiple of $1 + \alpha$ (note that the edge adjacent to the root is excluded).*

Below, we show that an optimal BMT always satisfies the edge length property and thus is embeddable (see Lemma 4). In Section 4, we will give an on-line algorithm whose schedule is represented by an embeddable BMT and show that the cost of this embeddable BMT is at most three times of any embeddable BMT, including those BMT representing the optimal schedule.

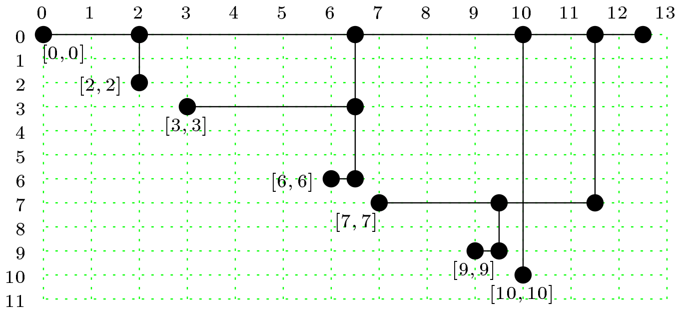


Figure 4: Suppose $\alpha = 1$. The rectilinear drawing in Figure 3(b) is embedded on a grid in which a horizontal grid arc has length $1 + \alpha = 2$ and a vertical grid arc has length $\alpha = 1$.

3.3 Optimal schedules

Notice that binary merge trees cannot represent all possible schedules. Yet in this section we show that under the assumption that the request sequence is compact, any optimal schedule can be represented by a binary merge tree that is embeddable.

First of all, we reveal two important properties induced by the above assumption. Roughly speaking, it means that an optimal schedule for a compact request sequence will merge all streams into one and each merging is binary in nature, i.e., no two streams merge with the same stream at the same time. Without loss of generality, we assume that the first stream is initiated at time 0. Recall that $K = \min\{\beta, \ell/(1+\alpha)\}$.

Lemma 3. *For any compact request sequence I, there is an optimal schedule O for I such that (i) every stream except the first stream merges with another stream at or before time $(1+\alpha)K$; and (ii) no two streams merge with another stream at the same time.*

Proof. Statement (i). Let s_1 denote the first stream. Suppose on the contrary that there exists a stream s_2 other than s_1 such that either s_2 merges with some stream after time $(1+\alpha)K$ or s_2 runs till the end without ever merging with any stream. Notice that if clients of s_2 start buffering s_1 at or before $(1+\alpha)K - \alpha\Delta_{s_1 s_2}$, where $\Delta_{s_1 s_2}$ denotes the difference between the initiation time of s_1 and s_2, s_2 can merge with s_1 at or before $(1+\alpha)K$ and the duration of s_2 can be shortened. Since O is optimal, the duration of each stream cannot be shortened. In other words, there exists a time $t_o > (1+\alpha)K - \alpha\Delta_{s_1 s_2}$ such that either (a) s_2 is initiated at t_o, or (b) there exists another stream s_3 merging with s_2 at t_o. If case (b) holds, clients of s_3, by definition, start buffering s_2 at $t_o - \alpha\Delta_{s_2 s_3}$. Then either (a) s_3 is initiated at $t_o - \alpha\Delta_{s_2 s_3}$ or (b) there exists another stream s_4 merging with s_3 at $t_o - \alpha\Delta_{s_2 s_3}$.

Repeating the above argument as long as case (b) holds, we can eventually unveil a sequence of merging events involving $m > 1$ streams $s_1, s_2, \cdots, s_m$ such that s_m is initiated at time $t_o - \sum_{i=2}^{m-1} \alpha\Delta_{s_i s_{i+1}} > (1+\alpha)K - \alpha\Delta_{s_1 s_2} - \sum_{i=2}^{m-1} \alpha\Delta_{s_i s_{i+1}} \geq (1+\alpha)K - \alpha K = K$. This contradicts that I is compact.

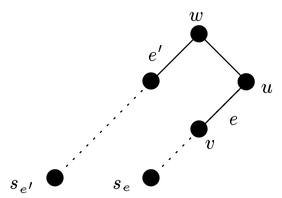
Statement (ii). Suppose on the contrary that with respect to O, two streams s_2 and s_3 merge with an earlier stream s_1 at time t. We assume that s_1, s_2 and s_3 are initiated respectively at t_1, t_2, and t_3, where $t_1 < t_2 < t_3$. By definition, clients of s_2 and s_3 start buffering s_1 at time $t - \alpha\,(t_2 - t_1)$ and $t - \alpha\,(t_3 - t_1)$, respectively. Both s_2 and s_3 terminate at t.

We modify O to give another schedule O' in which s_3 merges with s_2, followed by s_2 with s_1. At time $t - \alpha(t_3 - t_1)$, clients of s_3 start buffering s_2, and s_3 merges with s_2 at $t - \alpha(t_3 - t_1) + \alpha(t_3 - t_2) = t - \alpha(t_2 - t_1)$. Then clients of s_2 start buffering s_1, and s_2 merges with s_1 at $t - \alpha(t_2 - t_1) + \alpha(t_2 - t_1) = t$. Note that s_3 terminates at $t - \alpha(t_2 - t_1) < t$, while s_2 terminates at t. Therefore, the full cost of O' is less than that of O and a contradiction occurs. □

Given a schedule that satisfies the two properties stated in Lemma 3, we can construct the corresponding binary merge tree recursively starting from the last merging; the resulting tree has the same full cost and partial cost. Therefore, by Lemma 3, any optimal schedule for a compact request sequence can be represented by a binary merge tree. In other words, for any compact request sequence, the optimal schedule and the optimal binary merge tree have the same cost.

Next, we show that an optimal binary merge tree is G-embeddable. By Fact 2, it suffices to show that the duration of every left edge in the tree, except possibly the one adjacent to the root, is a multiple of $(1 + \alpha)$.

Lemma 4. *Let T be an optimal binary merge tree for a compact request sequence. For any internal node u of T, except possibly the root, (i) u has two sons; and (ii) the left edge e of u is associated with a duration (i.e., d_e) of $(1+\alpha)j$ time units for some integer j.*

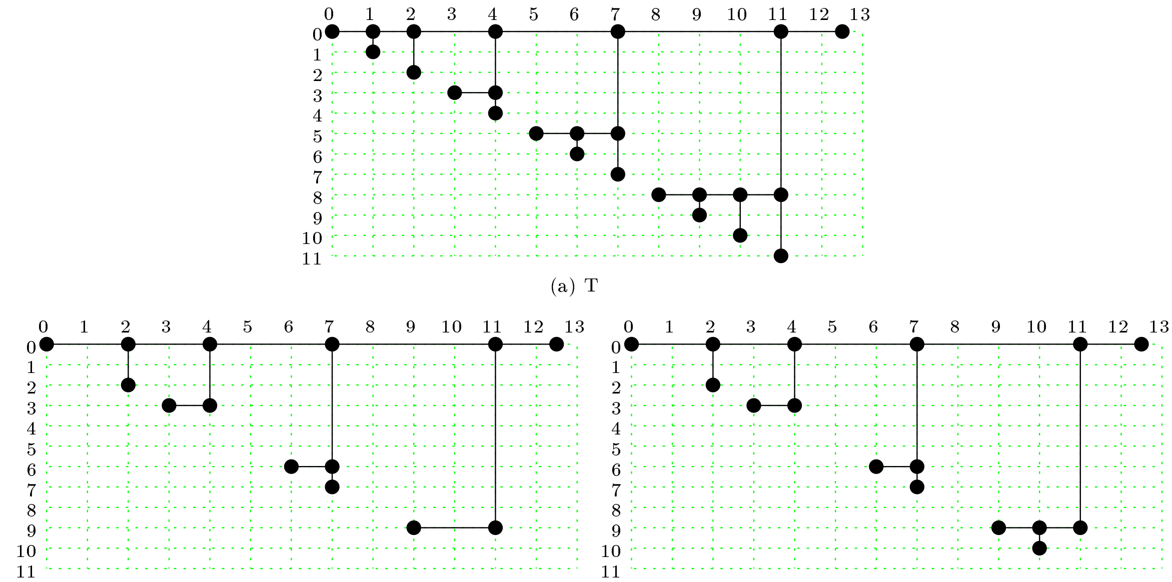


Proof. Statement (i). Suppose on the contrary that there is an internal node u other than the root which has only one son. By definition, u must have a left son, say v, and u is the right son of its parent, say w. Let e and e' be the left edge of u and w, respectively. Notice that s_e runs for $d(s_e, u)$ time units and then clients of s_e start buffering $s_{e'}$. We can modify T so that clients of s_e start buffering $s_{e'}$ after s_e runs for $d(s_e, v) < d(s_e, u)$ time units. The full cost of the resulting tree is less than that of T, contradicting the optimality of T.

Statement (ii). Denote v as the left son of u, and denote e and s_e as the edge (u, v) and its associated stream, respectively. Below we show that $d(s_e, u)$ is a multiple of $(1 + \alpha)$. Then $d(s_e, v)$ and thus $d_e = d(s_e, u) - d(s_e, v)$ is also a multiple of $(1 + \alpha)$.

Let $u_1, u_2, \ldots, u_m$ be the nodes along the rightmost path in the subtree rooted at u, i.e., $u_1 = u$ and each u_i is the right son of u_{i-1}. By Statement (i), $m > 1$ and u_m must be a leaf. Let s_i be the stream associated with the left edge of u_i, for $1 \leq i < m$, and let s_m be the stream associated with the leaf u_m. Let t_i be the initiation time of s_i. When s_2 merges with s_1, s_2 has played $d(s_2, u_1) = d(s_2, u_2) + \alpha(t_2 - t_1)$ time units and s_1 has played $d(s_1, u_1)$ time units. Thus, we have $t_1 + d(s_1, u_1) = t_2 + d(s_2, u_2) + \alpha(t_2 - t_1)$, or equivalently, $d(s_1, u_1) = d(s_2, u_2) + (1 + \alpha)(t_2 - t_1)$. Similarly, for $1 \leq i < m$, we have $d(s_i, u_i) = d(s_{i+1}, u_{i+1}) + (1 + \alpha)(t_{i+1} - t_i)$. Notice that $d(s_m, u_m) = 0$ because u_m is a leaf. Summing up all the equalities for $1 \leq i < m$, we have $d(s_1, u_1) = d(s_m, u_m) + (1 + \alpha)(t_m - t_1) = (1 + \alpha)(t_m - t_1)$. □

In conclusion, for any compact request sequence I, an optimal schedule can be represented by an embeddable BMT.



(a) T

(b) $\mathcal{A}_T(I_6)$

(c) $\mathcal{A}_T(I_7)$

Figure 5: (a) A complete EBMT T on the grid G. (b) $\mathcal{A}_T(I_6)$, where I_6 contains six requests arriving at time 0, 2, 3, 6, 7, and 9. (c) $\mathcal{A}_T(I_7)$, where I_7 contains the requests in I_6 and a new request arriving at time 10. Note that a_7 and b_7 are at grid points $[8, 10]$ and $[9, 10]$, respectively.

4 The on-line algorithm

4.1 Framework

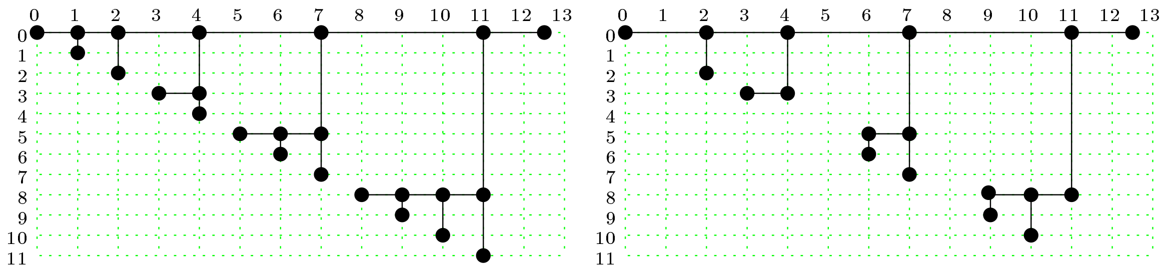
A crucial issue in designing on-line algorithms is to estimate the cost of the optimal solutions. In this paper, we show how to estimate the cost of the optimal schedules for *all* compact request sequences using one embeddable binary merge tree (EBMT), called a *complete* EBMT. This EBMT when restricted to any compact request sequence I gives a reasonable and attainable estimate of the cost of the optimal EBMT for I.

A *complete* EBMT T is any planar EBMT whose leaves comprise all grid points $[0, 0], \ldots, [K, K]$ along the diagonal of the grid G. See Figure 5(a) for an example. Below we show a simple way to make use of a given complete EBMT T to derive an on-line algorithm $\mathcal{A}_T$ for scheduling stream merging. For any compact request sequence I, $\mathcal{A}_T$ gives a schedule whose corresponding EBMT, denoted by $\mathcal{A}_T(I)$, has a smaller or the same partial cost as T when restricted to I (see Lemma 5). In Section 5, we construct a particular complete EBMT $\mathcal{T}$ such that the partial cost of $\mathcal{T}$ when restricted to any I is at most three times that of the optimal EBMT for I. This implies that $\mathcal{A}_{\mathcal{T}}$ is 3-competitive with respect to compact request sequences.

4.2 The on-line algorithm

Let T be a given complete EBMT. Based on T, we derive an on-line algorithm $\mathcal{A}_T$ for scheduling any compact request sequence as follows.

Recall that ℓ is the length of a full stream. When the first request arrives (at time 0), $\mathcal{A}_T$ initiates a stream for this request and this stream will not merge with any stream.



(a) T. (b) $T\|_I$.

Figure 6: (a) T. (b) $T\|_I$, where I is a compact request sequence containing seven requests arriving at time 0, 2, 3, 6, 7, 9, and 10.

I.e., the initial EBMT defined by $\mathcal{A}_T$ contains only two nodes on the grid G: a leaf at the top leftmost corner $[0,0]$ and the root at the point with length ℓ from $[0,0]$. These two nodes are connected by a horizontal grid line of length ℓ.

When the i-th request arrives, say, at time t_i, it is assigned a dedicated stream immediately. Let I_{i-1} denote the previous $i-1$ requests and let I_i denote I_{i-1} plus the i-th request. To determine which stream the i-th stream should merge with, we augment the current EBMT, i.e., $\mathcal{A}_T(I_{i-1})$, to include a new path for the leaf at $[t_i, t_i]$, resulting a new EBMT $\mathcal{A}_T(I_i)$. Let t_{i-1} be the arrival time of the $(i-1)$-st request. We locate in T the least common ancestor a_i of the leaves at $[t_i, t_i]$ and $[t_{i-1}, t_{i-1}]$ and add into $\mathcal{A}_T(I_{i-1})$ the following path for $[t_i, t_i]$.

Suppose a_i resides in $[r_i, c_i]$. The new grid path, denoted by σ_i, comprises a horizontal edge from the leaf at $[t_i, t_i]$ to a new node at $[t_i, c_i]$, and a vertical edge from $[t_i, c_i]$ up to the lowest grid point b_i where a horizontal edge e of the current EBMT is encountered. See Figures 5(b) and 5(c) for an example. Intuitively, this path means that after the i-th stream runs for $(1+\alpha)(c_i - t_i)$ time units, its clients start buffering from the stream associated with e. After another $\alpha(t' - t_i)$ time units, where t' is the row b_i lies, the i-th stream merges with the stream associated with e.

To analyze the performance of $\mathcal{A}_T$, we need to formalize the notion of tree restriction, which is defined with reference to the embedding of T on the grid G. We define the restriction of an EBMT T to a compact request sequence I, denoted by $T\|_I$, to be a tree on the grid G containing all the leaves associated with the requests in I, as well as all the edges and nodes along every path of T from a leaf in I to the root. See Figures 6(a) and 6(b) for an example. Note that $T\|_I$ is unique for any given I and may not be a BMT.

The next lemma gives an upper bound of the partial cost of $\mathcal{A}_T(I)$ for any compact request sequence I. It also reveals that $T\|_I$ is bounded below by $\mathcal{A}_T(I)$, i.e., for any point x on $T\|_I$, $\mathcal{A}_T(I)$ contains x or a point below it. Recall that $p(\mathcal{A}_T(I))$ denotes the partial cost of $\mathcal{A}_T(I)$. Let $p(T\|_I)$ be the total length of all the grid arcs in $T\|_I$ except those on the top row of the grid G.

Lemma 5. *For any complete EBMT T and compact request sequence I, (i) $p(\mathcal{A}_T(I)) \leq p(T\|_I)$; and (ii) $T\|_I$ is bounded below by $\mathcal{A}_T(I)$.*

Proof. Suppose I has k requests. We use induction to prove that for all $1 \leq i \leq k$, (i) $p(\mathcal{A}_T(I_i)) \leq p(T\|_{I_i})$; and (ii) $T\|_{I_i}$ is bounded below by $\mathcal{A}_T(I_i)$. When $i = 1$, these two statements are true. Suppose Statements (i) and (ii) are true for the first $i-1$ requests. We prove the case for i as follows.

Recall that σ_i, a_i, and b_i are defined in the above discussion. Furthermore, for any points x and y in T, we denote by $\sigma_T(x,y)$ the path between x and y in T. For any path σ, we denote by $|\sigma|$ the length of σ.

To prove Statement (i), we first observe that $T\|_{I_i}$ comprises $T\|_{I_{i-1}}$ and the path $\sigma_T([t_i,t_i],a_i)$. Let r_T be the root of T. Since T is a planar EBMT, the first point where the path $\sigma_T([t_i,t_i],a_i)$ intersects $T\|_{I_{i-1}}$ must be on the path $\sigma_T([t_{i-1},t_{i-1}],r_T)$, i.e., at the node a_i. In other words, edges in $\sigma_T([t_i,t_i],a_i)$ and $T\|_{I_{i-1}}$ correspond to disjoint lines on G, and $p(T\|_{I_i}) = p(T\|_{I_{i-1}}) + |\sigma_T([t_i,t_i],a_i)|$.

By construction, $\mathcal{A}_T(I_i)$ comprises $\mathcal{A}_T(I_{i-1})$ and σ_i, thus, $p(\mathcal{A}_T(I_i)) = p(\mathcal{A}_T(I_{i-1})) + |\sigma_i|$. By induction hypothesis, $p(\mathcal{A}_T(I_{i-1})) \le p(T\|_{I_{i-1}})$. Note that σ_i is the shortest rectilinear line joining $[t_i,t_i]$ and b_i. Since $T\|_{I_{i-1}}$ is bounded below by $\mathcal{A}_T(I_{i-1})$, b_i is equal to a_i or below a_i, and $|\sigma_i| \le |\sigma_T([t_i,t_i],a_i)|$. Thus, $p(\mathcal{A}_T(I_i)) \le p(T\|_{I_{i-1}}) + |\sigma_T([t_i,t_i],a_i)| = p(T\|_{I_i})$, and Statement (i) follows. To prove Statement (ii), observe that $\sigma_T([t_i,t_i],a_i)$ is bounded below by σ_i, and hence $T\|_{I_i}$ is bounded below by $\mathcal{A}_T(I_i)$. □

5 A good complete EBMT

In this section we describe a complete EBMT $\mathcal{T}$ such that $p(\mathcal{T}\|_I) \le 3p(Z_I))$ for any compact request sequence I, where Z_I denotes the optimal EBMT for I. Together with Lemma 5, we have $p(\mathcal{A}_\mathcal{T}(I)) \le p(\mathcal{T}\|_I) \le 3p(Z_I)$.

Before we give the details of $\mathcal{T}$, we observe that Z_I lies on a particular region of G. Recall that $K = \min\{\beta, \ell/(1+\alpha)\}$. Let G_0 be the subgrid of G bounded between row 0 and row K, and between column 0 and column K. We say that G_0 is a $(K+1)\times(K+1)$ grid. By Lemma 3(i), we know that with respect to Z_I, every stream except the first one merges with an earlier stream at or before time $(1+\alpha)K$. This implies that the first stream of Z_I corresponds to a horizontal grid line of length ℓ starting from $[0,0]$ and the lines for all the other streams lie on G_0.

To construct $\mathcal{T}$, we first construct a subtree $\mathcal{T}_o$ lying on G_0 as follows. Let $g(n) = \lfloor((1+\alpha)/(1+2\alpha))n\rfloor + 1$. The root of $\mathcal{T}_o$ is at the grid point $[0,K]$ of G_0. All the $K+1$ grid points along the diagonal are the leaves of $\mathcal{T}_o$. If $\mathcal{T}_o$ has only one leaf, we are done. If $\mathcal{T}_o$ has at least two leaves, it has two subtrees. The left subtree has its root at $[0, g(K)-1]$ and it contains the top $g(K)$ leaves along the diagonal. The right subtree has its root at $[g(K), K]$ and it contains the rest $K+1-g(K)$ leaves. These two subtrees are then constructed recursively, and their roots are connected to the root of $\mathcal{T}_o$ by a horizontal grid line with $K+1-g(K)$ grid arcs and a vertical grid line with $g(K)$ grid arcs. The EBMT $\mathcal{T}$ is obtained from extending $\mathcal{T}_o$ by a horizontal line connecting the point $[0,K]$ and the point with length ℓ from $[0,0]$. See Figure 7 for an example.

In the rest of this section, we show that $p(\mathcal{T}\|_I) \le 3p(Z_I)$. The key idea is to transform Z_I into another tree Y_I that is a subtree of $\mathcal{T}$ containing all the leaves of I. This transformation is conservative in replacing the edges of Z_I so as to ensure that $p(Y_I) \le 3p(Z_I)$, where $p(Y_I)$ is the total length of all grid arcs in Y_I except those at the top row of G. On the other hand, note that Y_I contains all the grid arcs in $\mathcal{T}\|_I$ and thus $p(\mathcal{T}\|_I) \le p(Y_I)$. We conclude that $p(\mathcal{T}\|_I) \le p(Y_I) \le 3p(Z_I)$.

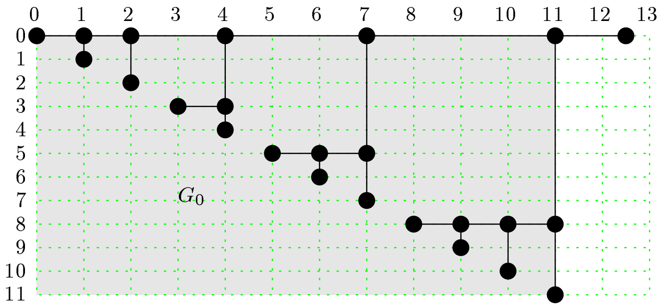


Figure 7: The EBMT $\mathcal{T}$ (assuming $\alpha = 1$). The shaded region is the grid G_0 which is of size 12×12. $\mathcal{T}_o$ is the subtree of $\mathcal{T}$ rooted at $[0, 11]$.

5.1 The recursive transformation

In this section we describe a procedure called TRANSFORM that transforms Z_I into a tree Y_I, which uses grid arcs of $\mathcal{T}$ only and whose set of leaves containing I. In the next section we prove that $p(Y_I) \leq 3p(Z_I)$.

Let Z_0 be the part of Z_I on G_0. We add to Z_0 a node at the top rightmost corner of G_0 (if it is not already there), which becomes the root of Z_0. From Lemma 4(i), we conclude that all internal nodes, except possibly the root, of Z_0 have two sons. Procedure TRANSFORM transforms Z_0 into a tree Y_0 with the same root. The tree Y_I is obtained from extending Y_0 by adding a horizontal line connecting the roots of Y_0 and Z_I.

Procedure TRANSFORM is recursive in nature; it takes two parameters: a grid and a tree. Initially, the grid is G_0 and the tree is Z_0. To explain the first step of TRANSFORM, we consider the rectangle R whose top rightmost corner is at $[0, K]$ and bottom leftmost corner is at $[g(K), g(K)-1]$. According to the way $\mathcal{T}$ is constructed, $\mathcal{T}$ does not *cross* R, i.e., there is no path in $\mathcal{T}$ entering R at a certain boundary grid point of R, going through at least one of its interior grid arcs, and leaving R at a distinct boundary grid point. See Figure 8(a) for an example. Thus, if Z_0 crosses R, the transformation process must remove the corresponding grid arcs within R. Details are as follows.

If Z_0 contains a path crossing R from the bottom boundary of R, we augment Z_0 to include all the grid arcs on the bottom and right boundaries of R, except the bottom leftmost grid arc of R. Similarly, if Z_0 contains a path crossing R from the left boundary of R, we include grid arcs on the left and top boundaries of R, except the left bottommost grid arc of R. Note that all newly added arcs are in $\mathcal{T}$. Afterwards, we remove all grid arcs of Z_0 that are within R. See Figure 8(b) for an example. Note that new intersections of grid arcs are probably introduced and new nodes will be created at these points.

Consider the $g(K) \times g(K)$ grid G_1 with top rightmost corner at $[0, g(K)-1]$ and the $(K+1-g(K)) \times (K+1-g(K))$ grid G_2 with top rightmost corner at $[g(K), K]$. See Figure 8(c) for an example. Notice that Z_0 lies in $G_1 \cup G_2 \cup R$ and the above step removes all arcs of $Z_0 - \mathcal{T}$ in R. Those arcs of $Z_0 - \mathcal{T}$ in G_1 and G_2 are removed recursively as follows. Let Z_0' denote the new tree formed after the above step. Below, we explain how to handle the part of Z_0' on G_1. Observe that all paths in Z_0' starting from the leaves on G_1 to the root of Z_0' exit G_1 through its top rightmost corner. If this corner is not a node of Z_0', we add a node there. Then, the part of Z_0' on G_1 forms a tree Z_1 that is on G_1 with the root at the top rightmost corner of G_1. If G_1 is of size at least 3×3

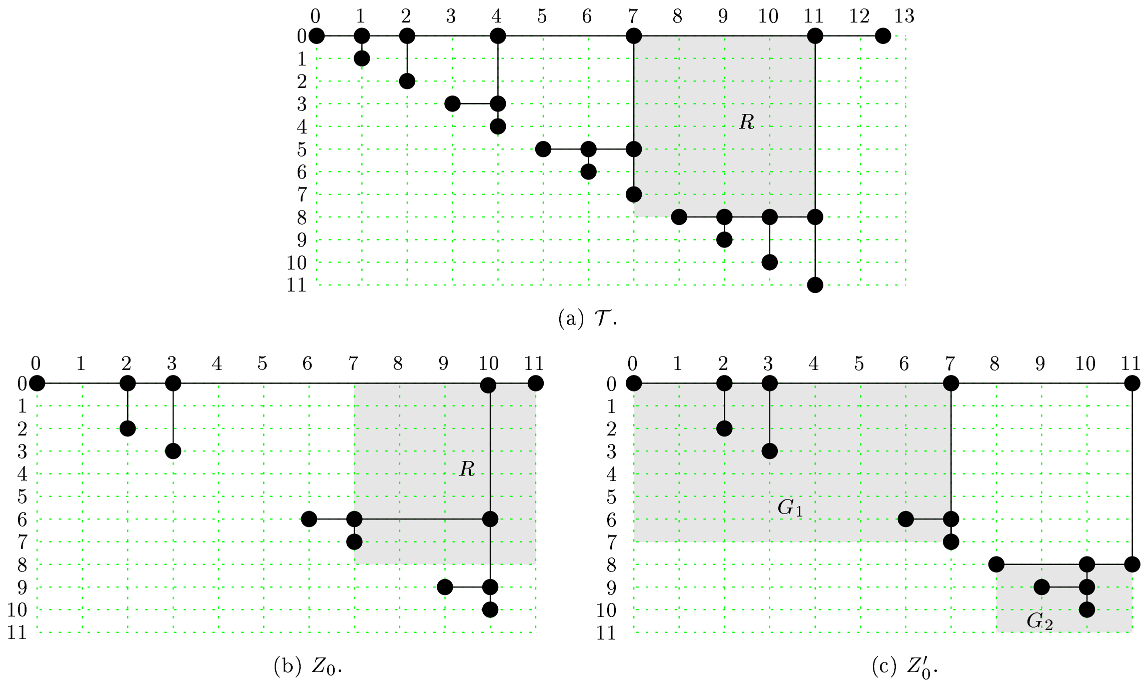


Figure 8: (a) $\mathcal{T}$. The rectangle R has top rightmost corner at $[0, 11]$ and bottom leftmost corner at $[8, 7]$. (b) Z_0. The grid arcs of Z_0 within R should be removed and those on the boundaries should be included. (c) Z_0', the 8×8 grid G_1 and the 4×4 grid G_2.

and Z_1 contains at least a leaf, we recursively call Procedure Transform on G_1 and Z_1 to transform Z_1 to a tree Y_1 that uses only grid arcs of $\mathcal{T}$ and retains the corresponding leaves. We handle the part of Z_0' on G_2 similarly, and let Y_2 be the resulting tree. Then, Y_1 and Y_2, together with the tree edges of Z_0' on the boundaries of R, forms Y_0.

5.2 Analysis of the recursive transformation

Let $p(Y_0)$ and $p(Z_0)$ be respectively the total length of lines in Y_0 and Z_0 except those on the top row of G_0. Note that $p(Y_0) = p(Y_I)$ and $p(Z_0) = p(Z_I)$. In this section we prove that $p(Y_0) \leq 3p(Z_0)$ and thus $p(Y_I) = p(Y_0) \leq 3p(Z_0) = 3p(Z_I)$.

Let E_a and E_r be the set of grid arcs added and removed by calling Transform on the grid G_0 and the tree Z_0. For any set of grid arcs E, we denote by $|E|$ the total length of grid arcs in E. A crucial step in our analysis is to show that $|E_a| \leq 3|E_r|$. Note that it suffices to show that in each recursive step (excluding the recursive calls), the total length of grid arcs added is at most three times the total length of grid arcs removed. Since the analysis is the same for every recursive step, we only consider the first step of Procedure Transform, in which the grid arcs in Z_0 within R are removed, where R is the rectangle in G_0 whose top rightmost and bottom leftmost corners are at $[0, K]$ and at $[g(K), g(K) - 1]$, respectively. Let $\mathcal{E}_a$ and $\mathcal{E}_r$ denote the set of grid arcs removed and added in this particular step, respectively. To prove $|\mathcal{E}_a| \leq 3|\mathcal{E}_r|$, we need the following lemma.

Lemma 6. *If Z_0 crosses R, Z_0 contains a path crossing R vertically or horizontally, i.e., the path passes through a single column or a single row of R.*

Proof. Suppose Z_0 has a path σ from a leaf to the root that crosses R. The path σ leaves R through either the right boundary or the top boundary. If σ leaves R through the right boundary, say on row i, σ or another path in Z_0 must go through R horizontally on row i. Otherwise, there exists a grid arc on row i not involved in Z_0; we assume the rightmost such grid arc occurs between the grid points $[i, x]$ and $[i, x+1]$. Note that $[i, x+1]$ corresponds to a node with at most one son. However, from Lemma 4(i), we conclude that all internal nodes, except possibly the root, of Z_0 have two sons. A contradiction occurs. If σ leaves R through the top boundary, say on column j, a similar argument can show that σ or another path in Z_0 must go through R vertically on column j. Therefore, the lemma follows. □

If Z_0 does not cross R, no grid arcs are added or removed and obviously $|\mathcal{E}_a| \leq 3|\mathcal{E}_r|$. Otherwise, we consider two cases, depending on whether Z_0 has a path crossing R vertically or horizontally. Recall that R has $g(K)+1$ rows and $K+2-g(K)$ columns. Let $h(K)$ and $w(K)$ denote the length of the right and top boundaries of R, respectively. That is, $h(K) = \alpha g(K)$ and $w(K) = (1+\alpha)(K+1-g(K))$.

Lemma 7. *If Z_0 contains a path crossing R vertically, then $|\mathcal{E}_r| \geq h(K)$ and $|\mathcal{E}_a| - |\mathcal{E}_r| \leq h(K) + w(K) - 1 - 2\alpha$. If Z_0 contains a path crossing R horizontally, then $|\mathcal{E}_r| \geq w(K)$ and $|\mathcal{E}_a| - |\mathcal{E}_r| \leq h(K) + w(K) - 1 - 2\alpha$.*

Proof. We only prove the first scenario, and the second one can be proved symmetrically. Let $H_{\mathtt{top}}$ and $H_{\mathtt{in}}$ be the sets of horizontal grid arcs in Z_0 that are on the top boundary of R and inside R, respectively. Besides the horizontal grid arcs, Z_0 has at least one whole column of grid arcs inside R (see Lemma 6). Thus, $|\mathcal{E}_r| \geq h(K) + |H_{\mathtt{in}}| \geq h(K)$. To estimate $|\mathcal{E}_a| - |\mathcal{E}_r|$, we consider two cases.

Suppose Z_0 has no path crossing R from the left boundary. Procedure Transform only expands Z_0 to include the bottom and right boundaries. Excluding the bottom leftmost grid arc of R, we have $|\mathcal{E}_a| \leq h(K)+w(K)-1-\alpha$. Thus, $|\mathcal{E}_a|-|\mathcal{E}_r| \leq w(K)-1-\alpha$.

Suppose Z_0 has a path crossing R from some grid point x on the left boundary of R. Procedure Transform will expand Z_0 to include all four boundaries, excluding the two grid arcs adjacent to the bottom leftmost corner. Note that the total length of these two grid arcs is $1+2\alpha$ and the length of the newly added grid arcs that are on the top boundary is $w(K) - |H_{\mathtt{top}}|$. Thus, $|\mathcal{E}_a| \leq 2h(K) + w(K) + (w(K) - |H_{\mathtt{top}}|) - 1 - 2\alpha$. Consider the path σ from x to the root of Z_0. Obviously, the total length of horizontal grid arcs in σ is at least $w(K)$, and is at most $|H_{\mathtt{top}}| + |H_{\mathtt{in}}|$. This implies $w(K) \leq |H_{\mathtt{top}}| + |H_{\mathtt{in}}|$, or equivalently, $w(K) - |H_{\mathtt{top}}| - |H_{\mathtt{in}}| \leq 0$. Therefore, $|\mathcal{E}_a| - |\mathcal{E}_r| \leq 2h(K) + w(K) + (w(K) - |H_{\mathtt{top}}|) - 1 - 2\alpha - (h(K) + |H_{\mathtt{in}}|) \leq h(K) + w(K) - 1 - 2\alpha$.

In both cases, $|\mathcal{E}_a| - |\mathcal{E}_r| \leq h(K) + w(K) - 1 - 2\alpha$. The lemma follows. □

The following technical lemma gives the relationships between $h(K)$ and $w(K)$.

Lemma 8. *(i) $h(K) \geq w(K) - 1 - \alpha$; and (ii) $w(K) \geq h(K) - \alpha$.*

Proof. Recall that $g(K) = \lfloor (1+\alpha)K/(1+2\alpha) \rfloor + 1$, $h(K) = \alpha g(K)$ and $w(K) = (1+$

$\alpha)(K+1-g(K))$. Then we have

$$\begin{aligned} h(K) &\geq \alpha(1+\alpha)K/(1+2\alpha) \\ &\geq (1+\alpha)(\lceil \alpha K/(1+2\alpha) \rceil - 1) \\ &= (1+\alpha)(K - \lfloor (1+\alpha)K/(1+2\alpha) \rfloor - 1) \\ &= w(K) - 1 - \alpha. \end{aligned}$$

On the other hand, we have

$$\begin{aligned} w(K) &= (1+\alpha)(K+1-\lfloor (1+\alpha)K/(1+2\alpha) \rfloor - 1) \\ &= (1+\alpha)(\lceil K - (1+\alpha)K/(1+2\alpha) \rceil) \\ &\geq \alpha\,(1+\alpha)K/(1+2\alpha) \\ &\geq h(K) - \alpha. \end{aligned}$$

Therefore, the lemma follows. □

Together with Lemma 7, we have $|\mathcal{E}_a| - |\mathcal{E}_r| \leq 2\,|\mathcal{E}_r|$, i.e., $|\mathcal{E}_a| \leq 3\,|\mathcal{E}_r|$. Therefore, we have $|E_a| \leq 3\,|E_r|$. We further prove that $p(Y_0)$ is at most three times $p(Z_0)$.

Lemma 9. $p(Y_0) \leq 3p(Z_0)$.

Proof. Note that E_a and E_r are disjoint, and thus $|Y_0| = |Z_0| + |E_a| - |E_r|$. Since all the grid arcs on the top boundary of G_0 are in Z_0 and Y_0, we have $p(Z_0) = |Z_0| - x$ and $p(Y_0) = |Y_0| - x$, where x is the length of the top boundary of G_0. Therefore, $p(Y_0) = p(Z_0) + |E_a| - |E_r|$. Since we only remove the grid arcs in Z_0 and all grid arcs on the top boundary of G_0 are not removed, $|E_r| \leq p(Z_0)$. Therefore, $p(Y_0) \leq p(Z_0) + 2|E_r| \leq 3p(Z_0)$. □

By the above discussion, we have the following theorem.

Theorem 10. *Given any compact request sequence I, we have $p(\mathcal{A}_{\mathcal{T}}(I)) \leq 3\,p(Z_I)$.*

Proof. By Lemma 5, we have $p(\mathcal{A}_{\mathcal{T}}(I)) \leq p(\mathcal{T}\|_I)$. Recall that $p(\mathcal{T}\|_I) \leq p(Y_I)$. By Lemma 9, we have $p(Y_I) = p(Y_0) \leq 3\,p(Z_0) = 3\,p(Z_I)$. Therefore, we have $p(\mathcal{A}_{\mathcal{T}}(I)) \leq 3\,p(Z_I)$ and the theorem follows. □

6 General request sequences

In this section, we show how to generalize $\mathcal{A}_{\mathcal{T}}$ to handle an arbitrary request sequence J. Then, by adapting the analysis of Bar-Noy and Ladner [4], we prove that the generalized algorithm is 5-competitive.

We can use $\mathcal{A}_{\mathcal{T}}$ to process an arbitrary request sequence J as follows. As requests of J arrive, we divide them into groups as follows. When a request arrives within $K = \min\{\beta, \ell/(1+\alpha)\}$ time units from the first request of the current group, we insert it to this group, initiate a new stream for it, and use $\mathcal{A}_{\mathcal{T}}$ to determine how this stream merges with another stream in the group. Otherwise, we start a new group and initiate a full stream for this request. Effectively, we divide J into maximal and non-overlapping compact request sequences $C_1, C_2, \ldots$ and use $\mathcal{A}_{\mathcal{T}}$ to find a schedule for each C_i. The schedule for J, $\mathcal{A}_{\mathcal{T}}(J)$, is just the union of these schedules. Below we show that this generalized $\mathcal{A}_{\mathcal{T}}$ is 5-competitive.

Recall that for any schedule S, $f(S)$ denotes the full cost of S. Consider any optimal schedule OPT_J for serving arbitrary request sequence J. The following lemma derives a lower bound on $f(OPT_J)$.

Lemma 11. *Suppose J can be divided into m maximal and non-overlapping compact request sequences. Then $f(OPT_J) \geq m(\ell - K)$.*

Proof. Suppose J is divided into maximal and non-overlapping compact request sequences $C_1, C_2, \ldots, C_m$. For $1 \leq i \leq m$, let q_i be the first request in C_i, and let $\mathcal{L}_i$ be the set of streams serving requests in $C_1 \cup C_2 \cup \cdots \cup C_{i-1} \cup \{q_i\}$. It suffices to prove that for $1 \leq i \leq m$, there are at least i streams in $\mathcal{L}_i$ that run for at least $\ell - K$ time units.

Since the first request of J is in C_1 and it must be served by a full stream, the lemma is true for $\mathcal{L}_1$. Suppose the lemma is true for $\mathcal{L}_{i-1}$. Let s be the last stream in $\mathcal{L}_i$, i.e., the one serving q_i. If there is a stream in $\mathcal{L}_i - \mathcal{L}_{i-1} - \{s\}$ that runs for at least $\ell - K$ time units, then the lemma is true for $\mathcal{L}_i$. Otherwise, we argue that s runs for at least $\ell - K$ time units as follows.

Suppose s is initiated at time t. Note that all streams initiated before s (i.e., those in $\mathcal{L}_i - \{s\}$) would have terminated at time $t + \ell - K - 1$ because (1) we suppose all streams in $\mathcal{L}_i - \mathcal{L}_{i-1} - \{s\}$ runs for no more than $\ell - K - 1$ time units, and (2) all streams in $\mathcal{L}_{i-1}$ are initiated before time $t - K - 1$ (otherwise, the sequence obtained by inserting q_i to C_{i-1} is still compact). Therefore, to view the last $K + 1$ units of video, clients of s can only get data from s or the buffer. Since at time $t + \ell - K - 1$, clients of s have at most $\min\{\beta, (\ell - K - 1)/\alpha\} \leq K$ units of video in the buffer, they must get some data from s after time $t + \ell - K - 1$. This implies s runs for at least $\ell - K$ time units. $\square$

Now, we prove that $\mathcal{A}_{\mathcal{T}}$ is 5-competitive.

Theorem 12. *For any request sequence J, $f(\mathcal{A}_{\mathcal{T}}(J)) \leq 5f(OPT_J)$.*

Proof. Suppose J can be divided into m maximal and non-overlapping compact request sequences $C_1, C_2, \ldots, C_m$. For $1 \leq i \leq m$, let OPT_{C_i} be the optimal schedule for serving C_i. From Theorem 10, we have $p(\mathcal{A}_{\mathcal{T}}(C_i)) \leq 3p(OPT_{C_i})$. Below, we show that $\sum_{1 \leq i \leq m} f(OPT_{C_i}) \leq f(OPT_J) + m\ell$ and thus we have

$$f(\mathcal{A}_{\mathcal{T}}(J)) = \sum_{1 \leq i \leq m} (\ell + p(\mathcal{A}_{\mathcal{T}}(C_i))) \leq m\ell + 3 \sum_{1 \leq i \leq m} (f(OPT_{C_i}) - \ell) \leq m\ell + 3f(OPT_J).$$

Lemma 11 suggests $m \leq f(OPT_J)/(\ell - K)$ and thus $f(\mathcal{A}_{\mathcal{T}}(J)) \leq f(OPT_J)(\ell/(\ell - K) + 3)$. Since $K = \min\{\beta, \ell/(1 + \alpha)\}$ and $\alpha \geq 1$, we have $\ell/(\ell - K) \leq 2$ and the theorem follows.

It remains to prove the inequality $\sum_{1 \leq i \leq m} f(OPT_{C_i}) \leq f(OPT_I) + m\ell$. Let s_i be the stream that serves the first request in C_i. We modify OPT_J to a new schedule O as follows. We extend every s_i to a full stream, and then for any stream serving some request in C_i, if it merges with a stream initiated earlier than s_i, then we make it merge with s_i; the duration of this stream is shortened. Thus, $f(O) \leq f(OPT_J) + m\ell$. Notice that O is the union of m schedules, each for one C_i. Since OPT_{C_i} is the optimal schedule for C_i, we have $\sum_{1 \leq i \leq m} f(OPT_{C_i}) \leq f(O)$. As a result, we have $\sum_{1 \leq i \leq m} f(OPT_{C_i}) \leq f(OPT_J) + m\ell$. $\square$

7 Remarks

In this paper we study the on-line stream merging problem in a VOD system so as to minimize the total server bandwidth. We give a 5-competitive on-line algorithm for stream merging in the general setting. Recently, Coffman *et al.* [9] proposed the dyadic algorithm for the restricted setting where $\alpha = 1$ and $\beta = \ell/2$ and showed an almost tight upper bound on the average total bandwidth requirement when the inputs follow the Poisson Distribution. We observe some similarities between the dyadic algorithm and our algorithm, and we conjecture that the dyadic algorithm also has a constant competitive ratio for the restricted setting.

Both the dyadic algorithm and our algorithm focus on minimizing the total bandwidth used by the system. Another metric that worths consideration is the maximum server bandwidth over time. Very recently, Bar-Noy *et al.* [3] have evaluated empirically the performance of several on-line algorithms with respect to this metric; there is no competitive analysis though. An open question is to find an $O(1)$-competitive on-line algorithm with respect to the maximum server bandwidth over time.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 253–258, 1996.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal piggyback merging policies for video-on-demand systems. In *Proceedings of ACM Sigmetrics Conference*, pages 200–209, 1996.

[3] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam. Comparison of stream merging algorithms for media-on-demand. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 115–129, 2002.

[4] A. Bar-Noy and R. E. Ladner. Competitive on-line stream merging algorithms for media-on-demand. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 364–373, 2001.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[6] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 204–215, 1999.

[7] S. W. Carter and D. D. E. Long. Improving bandwidth efficiency of video-on-demand. *Computer Networks*, 31(1-2):99–111, 1999.

[8] T. Chiueh and C. Lu. A periodic broadcasting approach to video-on-demand service. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 162–169, 1995.

[9] E. Coffman, P. Jelenkovi, and P. Momcilovic. The dyadic stream merging algorithm. *Journal of Algorithms*. To appear. A preliminary version appears in WCW 2001.

[10] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Journal of Multimedia Systems*, 4(3):112–121, 1996.

[11] D. Eager, M. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proceedings of the Seventh ACM International Conference on Multimedia*, pages 199–202, 1999.

[12] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 206–215, 2000.

[13] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):742–757, 2001.

[14] L. Golubchik, J. C. S. Lui, and R. R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *ACM Journal of Multimedia Systems*, 4(3):140–155, 1996.

[15] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of the Sixth ACM International Conference on Multimedia*, pages 191–200, 1998.

[16] S. W. Lau, J. C. S. Lui, and L. Golubchik. Merging video streams in a multimedia storage server: Complexity and heuristics. *ACM Journal of Multimedia Systems*, 6(1):29–42, 1998.

[17] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In *Proceedings of the Ninth International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 44–55, 1999.

[18] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Journal of Multimedia Systems*, 4(3):197–208, 1996.