On-Line Stream Merging, Max Span, and Min Coverage

Wun-Tat Chan\textsuperscript{1}, Tak-Wah Lam\textsuperscript{2}*,
Hing-Fung Ting\textsuperscript{2}**, and Prudence W.H. Wong\textsuperscript{2}

\textsuperscript{1} Department of Computing, Hong Kong Polytechnic University, Hong Kong,
cswtchan@comp.polyu.edu.hk
\textsuperscript{2} Department of Computer Science, University of Hong Kong, Hong Kong,
{twlam,hfting,whwong}@cs.hku.hk

Abstract. This paper introduces the notions of span and coverage for analyzing the performance of on-line algorithms for stream merging. We show that these two notions can solely determine the competitive ratio of any such algorithm. Furthermore, we devise a simple greedy algorithm that can attain the ideal span and coverage, thus giving a better performance guarantee than existing algorithms with respect to either the maximum bandwidth or the total bandwidth. The new notions also allow us to obtain a tighter analysis of existing algorithms.

1 Introduction

A typical problem encountered in video-on-demand (VOD) systems is that many requests for a particular popular video are received over a short period of time (say, Friday evening). If a dedicated video stream is used to serve each request, the total bandwidth requirement for the server is enormous. To reduce the bandwidth requirement without sacrificing the response time, a popular approach is to merge streams initiated at different times (see e.g., 111039131611214).

Stream merging is based on a multicasting architecture and assumes that each client has extra bandwidth to receive data from two streams simultaneously. In such a system, a stream can run in two different states: normal state and exceptional state. A new stream $X$ is initially in normal state and all its clients receive one unit of video from $X$ in every time unit for immediate playback. Some time later, $X$ may change to the exceptional state and all its clients receive and buffer an extra of $1/\lambda$ unit of video from an earlier stream $Y$ in every time unit, where $\lambda$ is an integer parameter characterizing the extra bandwidth allowed for a particular VOD system. When $X$’s clients have buffered enough data from $Y$, they can synchronize with the playback of $Y$ and all clients of $X$ can switch to $Y$. At this time, $X$ can terminate, and $X$ is said to merge with $Y$. Note that such a merging reduces the total bandwidth requirement.

* 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-7045/02E
To support stream merging effectively, we need an on-line algorithm to decide how streams merge with each other. The performance of such an on-line algorithm can be measured rigorously using the competitive ratio, i.e., the worst-case ratio between its total bandwidth and the total bandwidth used in an optimal schedule. The literature contains a number of on-line stream merging algorithms, e.g., the greedy algorithm [3] (also called nearest-fit), the Dynamic Fibonacci tree algorithm [3], the connector algorithm [7], and the $\alpha$-dyadic algorithm [9]. The greedy heuristic is attractive because of its simplicity and ease of implementation; a stream simply merges to the nearest possible stream. Unfortunately, it has been shown to be $\Omega(n/\log n)$-competitive, where $n$ is the total number of requests [3]. The other three algorithms provide much better performance guarantee; in particular, the connector algorithm and the $\alpha$-dyadic algorithm are known to be 3-competitive [7,6]. Yet these algorithms are much more complicated than the greedy algorithm. The Dynamic Fibonacci tree algorithm is based on a data structure called Fibonacci merge tree, the connector algorithm needs to pre-compute a special reference tree to guide the on-line algorithm, and the $\alpha$-dyadic algorithm is recursive in nature.

In reality, it might make more sense for a stream merging algorithm to minimize the maximum bandwidth over time instead of the total bandwidth [2]. In [6], we consider the special case when the extra bandwidth parameter $\lambda$ is equal to one (i.e., a client can receive 1 unit of normal and 1 unit of extra data in one time unit) and show that with respect to the maximum bandwidth, the connector algorithm is 4-competitive and the $\alpha$-dyadic algorithm is 4-competitive when $\alpha$ is chosen to be 1.5. Empirical studies indeed confirm that the connector algorithm and the $\alpha$-dyadic algorithm do have very similar performance under different measurements [2,16].

As far as we know, the best lower bounds on the competitive ratios with respect to the maximum bandwidth and the total bandwidth are $4/3$ and 1.139, respectively. An obvious question is whether we can further improve the analysis of existing algorithms or come up with another algorithm with a better competitive ratio. With a deep thought, we want to identify the key elements in designing a good stream merging algorithm and to explain why the connector algorithm and the $\alpha$-dyadic algorithm have similar performance. In this paper we attempt to answer the above questions.

When designing a stream merging algorithm, there are two conflicting concerns in determining how long a stream should run before it merges. Obviously, we want to merge it with an earlier stream early enough so as to minimize the bandwidth requirement. Yet we also want a stream to run long enough so that more streams initiated later can merge with it. Good algorithms such as the connector algorithm and the $\alpha$-dyadic algorithm must be able to balance these two concerns properly. In this paper we show how these two concerns can be measured rigorously and more importantly, can be used to determine the competitive ratio of an algorithm. More precisely, we define the notions of span factor and coverage