Efficient and Flexible Parallel Retrieval using Priority Encoded Transmission

Ramaprabhu Janakiraman, Lihao Xu
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130-4899, USA
E-mail: {rama, lihao}@cse.wustl.edu

Abstract—Many applications, including web transfers, software distribution, video-on-demand, and peer-to-peer data downloads, require the retrieval of structured documents consisting of multiple components like images, video, and text. Large systems using these applications may be made more scalable by using efficient data distribution techniques like multicast, and by enabling clients to retrieve data from multiple servers in parallel.

In this paper we propose a new technique for parallel retrieval of structured documents from multiple servers using priority encoded transmission, which allows some subsets of a transmission to be reconstructed before others. We discuss the application of this technique to bulk and streaming media distribution, and provide performance results from trace-based simulations.

1. Introduction

Motivations

Content distribution networks improve the end-user experience for clients of large web sites by distributing content at multiple servers spread over different geographic locations in the network. They often serve a large and heterogeneous client population, and the content they provide consists of complex documents containing many components like HTML files, images, video, Flash animations, Java applets etc.

Such infrastructures are also useful in video-on-demand systems. These systems are bandwidth-intensive, and will benefit from load-balancing by multiple servers as well as from efficient multicast-based data distribution [1]. Furthermore, these systems are marked by a heterogeneous client population in which clients, depending on their resource constraints, are interested in different subsets of the data (base layers, enhancement layers), possibly with different delays before playout.

A similar situation is encountered when high-bandwidth clients download content from peer-to-peer systems. To get an aggregate download rate higher than the individual bottleneck rates of individual peers, the client would like to attempt a parallel download of from multiple peers. If the peers are themselves are in the process of downloading the document, then it may be partially present at one or more peers.

Finally, in large self-organizing web caches, documents may be partially cached by multiple servers in a decentralized manner, leading to potential overlap with the content of other servers.

Problem and existing solutions

The common theme in all these examples should now be clear: for scalability and fault-tolerance, content providers wish to distribute their data over multiple servers. Clients of these systems wish to improve their aggregate download rate by retrieving in parallel from multiple servers, but must ensure that they don't end up getting duplicates from multiple servers.

One way to do this is for the provider to replicate the content on all servers. The client could then request disjoint portions from each server to ensure non-conflicting downloads. In a large content distribution network, this can be a high-cost solution. In decentralized server systems like peer-to-peer systems, data placement on peers cannot be centrally controlled, and the download process must necessarily occur in an ad-hoc manner.

Yet another solution [2] is to encode documents using a long erasure-resilient code. The client can download enough *distinct* data items that will enable it to recover the original content. This approach, while significantly simplifying the coordination necessary for parallel downloads, has its own shortcomings: first, for large data files, the encoding and decoding costs can be prohibitive. Although efficient codes suitable for bulk data distribution have been proposed [3], they do not solve the more significant problem of *flexible access* to documents that have an inherent *structure*. For example, for movie files, storing video and sound-tracks in several languages separately allows clients to download the video and the sound-track of their choice. In video-on-demand applications, encoding segments separately allows some segments to be reconstructed before others, so that clients can *stream* the video while still retrieving later segments.

Proposed solution

In this paper, we study the problem of parallel retrieval from many servers, where each server might have a subset of the data required by the client. In this case, parts of the document may be present at multiple servers with disparate resource constraints. To achieve efficient downloads, a client has to *co-ordinate* its download process with each server in a manner that reconciles the servers' capabilities with its own download requirements.

We propose the use of *priority encoded transmission* (PET) [4] for parallel retrieval of documents. The basic idea is that each client should determine priorities for each part of the document it retrieves from each server. Servers encode their data according to these priorities and send them as messages to the client. Clients receiving messages in parallel from multiple servers can recover the whole document at the end of retrieval, or part-by-part according to some pre-determined schedule.

Our solution has the following nice features:

Flexibility: It can be used for unicast downloads by individual clients as well as asynchronous multicast dissemination by a collection of servers. It may be adapted for bulk as well as streaming distribution modes. It can be used unchanged when data is partitioned, or replicated (fully or partially), across multiple servers.

Efficiency: For a given document spread across a given set of

servers, it determines the *optimal* priorities for retrieval from each server.

Loss-resilience: This comes for "free" from the encoding process used, and is particularly useful in asynchronous multicast data dissemination applications.

We organize the discussion as follows: in the next section, we formalize the problem of parallel retrieval, and give a motivating example for why coordinated transmission can lead to faster performance. In section 3, we introduce the problem of assigning priorities to the transmissions of each server, and illustrate how to obtain an optimal solution as well as approximate solutions with smaller computational and communication complexity. In section 5, we describe the stateless priority encoding process performed by the servers. We discuss the application of this method to both bulk and streaming data distribution in section 6, and provide results on their performance in section 7. Conclusions and a discussion of ongoing work end the paper.

2. Parallel retrieval

Keeping in mind the wide range of these applications, we are motivated to abstract away the details of individual applications, but instead come up with an idealized problem setting and a solution that is useful in a wide range of scenarios.

Abstract model

In this idealized setting, *symbols* are data units of fixed size. *Pages* are disjoint sets of symbols. *Documents* are (not necesarily disjoint) sets of pages. Pages are transmitted on *channels*. A channel C has a rate $|C|$ symbols per unit time. A page of P of size $|P|$ transmitted with *rate* $r \leq |C|$ on C can be retrieved in time $\frac{|P|}{r}$ by a client listening on the channel, irrespective of other pages transmitted on C.

A client in this system is described by the tuple $\langle D, C, \rho \rangle$, where D is a document the client wishes to download, C is a channel available to the client, and ρ is a *schedule* for the pages in D, which maps each page $P \in D$ to a rate $\rho(P)$ on C such that $\sum_{P \in D} \rho(P) \leq |C|$.

A server is described by $\langle X, C \rangle$, where X is a set of pages contained by the server, and C is the channel available to the server.

The download process for a client $\langle D, C, \rho \rangle$ proceeds in three phases: *location*, *initiation*, and *transmission*.

First it locates a set of servers from which to download D. We do not concern ourselves with how a client can locate servers on the network, which is a well-researched topic in its own right. Rather we assume that some mechanism such as described in e.g. [5] is available by which a client can locate enough servers whose combined output will enable it to download D in its entirety.

Assume that in the previous step, the client has located a set of servers $S = \{\langle X_i, C_i \rangle : i = 1 \text{ to } |S|\}$. We will focus on the download of a single document by a single client, so assume that $X_i \subset D$ and $\cup_{i=1}^{|S|} X_i = D$, and that each server S_i can transmit to the client at its channel rate $|C_i|$.

During *initiation*, each server S_i sends a description $\langle L_i, |C_i| \rangle$ to the client, where L_i enumerates its pages, i.e, $j \in L_i$ whenever $P_j \in X_i$. The client gathers descriptions from all servers. Using the descriptions $\langle L_i, |C_i| \rangle$ and its download schedule ρ, it computes a rate vector $R = \{r_{ij}\}$, where $S_i \in S$ and $P_j \in X_i$ and sends a request $R_i = \{(j, r_{ij})\}$, where $r_{ij} \in R$ to each server S_i.

During *transmission*, each server S_i, upon receiving a request R_i, transmits a message M_i on C_i to the client. Each message is a set of *packets*, and each packet is a set of symbols. By receiving packets from all servers, the client achieves its download schedule ρ for D.

Mapping to physical systems

Our abstract formulation is suitable for modeling a wide range of applications. For example, in a video-on-demand system, D is a movie file and each page is a contiguous segment of frames. For web content distribution, D is a complex web document and each page is an image or video clip etc. For a software update scheme, the document is (say) a new service pack, and each page represents a patch for an individual application. In the simplest case, D is just a data file and the pages comprise blocks of bits.

The channels can also be suitably interpreted: In a simple parallel download scheme, each server's channel C_i represents its unicast connection to the client, while the client's channel C is the aggregation of unicast connections with servers (In this case, the channel rate of each server is merely the bottleneck bandwidth of its unicast path to the client). In an asynchronous multicast scheme, the channel is a shared multicast session in which servers will transmit pages continuously, and clients will subscribe to the session, unsubscribing upon downloading the entire document. Heterogeneous clients can be supported by layering over multiple multicast sessions, in which case the client's channel is the aggregation of all the multicast groups to which it subscribes.

Problems to be solved

In the rest of this paper, we will focus on the *initiation* and *transmission* phases. In implementing the above protocol, we encounter two issues:

Rate allocation: During initiation, given descriptions $\langle L_i, |C_i| \rangle$ from each server, and its schedule ρ, the client should compute a rate vector R that will enable it to meet its download goals.

Information-additive transmission: During transmission, each server S_i, by sending an appropriately encoded message M_i of its content X_i, should enable the client to gather messages from all servers and fulfill its download schedule ρ for the entire document D.

To get a feel for the problem, consider the following example:

Example 1 *D is an 6 megabyte file with three pages P_1, P_2, P_3 each of size 2 megabytes. Server S_1 with bottleneck bandwidth $|C_1| = 200kbps$ has pages $P_1$and P_2, while S_2 with $|C_2| = 300kbps$ has P_2 and P_3.*

A client $\langle D, C \rangle$ with $|C| = 500kbps$ attempts to download D from both S_1 and S_2. Assume that it wishes to download D in the shortest time. Since its aggregate reception bandwidth is $500kbps$, it should set up a schedule $\rho = \{500/3, 500/3, 500/3\}$, to ideally finish downloading D in 96 seconds. If S_1 and S_2, oblivious to each other, both split their bandwidths equally between their pages, the aggregate download rates perceived by the client are $100kbps$ for P_1, $250kbps$ for P_2 and $150kbps$ for P_3. The entire download time, is 160 seconds. If instead it sends request $R_1 = \{(1, 500/3), (2, 100/3)\}$ to S_1 and $R_2 = \{(2, 400/3), (3, 500/3)\}$ to S_2, so that S_1 allocates $\frac{5}{6}$ of its bandwidth to P_1 and $\frac{1}{6}$ to P_2, and S_2 allocating $\frac{4}{9}$ of its bandwidth to P_2 and $\frac{5}{9}$ to P_1, then all pages can finish downloading at the optimal time of 96 seconds, meeting the schedule ρ. ■

3. Rate allocation

In the previous example, it was observed that by suitably allocating rates for different pages, the client can speed up its parallel download process. Intuitively, we can surmise that pages present in more servers will be assigned lower rates than those present only in a few.

Optimal solution

We now solve the general version of the problem, i.e., a client $\langle D, C, \rho \rangle$ allocating rates for each page $P_i \in D$ to each server $S_i \in S$ to meet its download schedule ρ. This can be formulated as an instance of the well-known *transportation* problem described succintly using a bipartite graph as follows:

1) There are $|S|$ *sources* corresponding to servers, such that source S_i can supply a quantity $|C_i|$. Similarly there are $|D|$ *sinks* corresponding to pages, such that sink P_i has a demand $\rho(P_i)$.
2) A source can supply to only a subset of sinks, and this is determined by a set of edges $E \subset \{1..|S|\} \times \{1..|D|\}$ connecting sources to sinks; $(i, j) \in E$ iff source S_i can supply to sink P_j i.e., $P_j \in X_i$.

Assume that an *allocation* $X : E \rightarrow \Re^+$ consists of S_i sending flow r_{ij} to P_j when $(i, j) \in E$. For such an allocation, let the total *outflow* from S_i be $\sigma_i = \sum_{(i,j)\in E} r_{ij}$ and let the *inflow* into P_j be $\rho_j = \sum_{(i,j)\in E} r_{ij}$.

The problem consists of determining a set of flows $r_{ij}, (i, j) \in E$ such that the inflow at any sink P_i (the aggregate download rate of page P_i from all servers) meets the demand $\rho(P_i)$ (the download rate required for P_i by the schedule) without the outflow at any source S_i (the rate of transmission at server S_i) exceeding its supply $|C_i|$ (the channel bandwidth of S_i). This can be easily formulated and solved as a linear programming problem to get a rate assignment that meet the client's schedule.

Feasible solutions

It can be observed that demand $\rho(P_j)$ at each sink P_j cannot always be met through its inflow ρ_j while simultaneously restricting the outflow σ_i at each source S_i to be no greater than its supply $|C_i|$. This motivates us to define an allocation in terms of an *objective function*, and seek for a *feasible* allocation that minimizes this objective.

In this case, the supply at each source (the transmission rate of each server) is a strict limit, potentially causing inflow to fall below demand (some pages to download later than the scheduled time) at some sinks. Assume that sink P_j will incur a cost $C_{P_j}(\rho_j)$ on an inflow ρ_j to meet its demand $\rho(P_j)$. The optimization problem in this case is to find an allocation that minimizes the maximum cost incurred by any sink:

$$\begin{aligned} &\text{Minimize} \max\ C_{P_j}(\rho_j) \quad j = 1, 2, \cdots, |D|, \\ &\text{Subject to constraints } \sigma_i \leq |C_i| \quad i = 1, 2, \cdots, |S| \end{aligned} \tag{1}$$

This problem can be formulated as a *linearly constrained minimax optimization*, which can be performed numerically using existing software packages [6].

Parameters

To solve concrete instances of the problem, we have to first determine 1) channel bandwidths of the servers $|C_i|$, 2) the placement of pages on servers E, 3) the schedule of the client ρ, and 4) the cost functions $C_{P_j}(\rho_i)$. The first two are related to the network, and the next two to the application. In section 6, we illustrate the application of this method to *bulk* and *streaming* data distribution.

4. Approximate solutions

Given cost functions $C_{P_j}(\rho_j)$, the server rates σ_i, and the client schedule $\rho(P_j)$, an optimal solution can always be found using minimax optimization. However this approach quickly becomes infeasible for large numbers of servers and pages. Hence, we are motivated to seek for efficient heuristics that can return an acceptable solution to the problem much faster. In this section, we propose such heuristics for reducing both computational and communication complexity; in section 7, we provide experimental results on their performance.

Reducing computational complexity

The basic idea of rate allocation is to favor those pages that are *scarce* over those that are *common*. Instead of trying to formulate this as an optimization problem, we try to capture this idea with the following notion of scarcity:

Definition 1
The *scarcity* of a page is the ratio of requested rate of the page to the aggregate rates of the server providing it.

$$\text{scarcity}(P_j) = \frac{\rho(P_j)}{\sum_{P_j \in X_i} |C_i|} \tag{2}$$

■

The heuristic we propose is simply: To determine a rate allocation, split the channel capacity of each server between its symbols in the ratio of their scarcities:

$$r_{ij} = |C_i| \frac{\text{scarcity}(P_j)}{\sum_{P_k \in X_i} \text{scarcity}(P_k)}, \text{ for all } P_j \in X_i \tag{3}$$

In section 7, we compare the performance of this heuristic against that of the optimal allocation described in the previous section.

Reducing communication complexity

We now study the *communication* complexity during the initiation phase. Each description of a server S_i contains a list of all pages $X_i \subset D$ on the server. If each server sends an enumeration of the pages as a description and the client responds by sending a request to each server as an enumeration of the requested per-page rates, the communication complexity $O(X_i)$ per server and $O(D \times S)$ as a whole. This can be significant for large documents. More efficient communication can be achieved at the cost of a little efficiencty with a more compact representation of enumerations by using *Bloom Filters* [7] to send descriptions and requests. Specifically, to represent a set of counters compactly, a *counting Bloom filter* [7] is used.

We propose the use of Bloom filters in the following way: Each server constructs a description by inserting the indices of its set of pages X_i into a counting Bloom filter, and sends this description to the client. The client uses these filters to compute its rate vector. To each server S_i, it sends a request which represents the set of rates $\{r_{ij} : P_j \in X_i\}$ as a counting Bloom filter. The server uses this to estimate a set of rates for its pages, and proceeds with the transmission according to these estimated rates.

5. Information-additive transmission

In this section, we discuss the primary issue in the transmission phase: each server S_i, upon receiving a rate vector R_i from the client, should send a message M_i to the client, which should enable the client to meet its schedule ρ, notwithstanding that some of the pages may be duplicated at multiple servers.

Naïve solution

A straightforward approach is to schedule transmissions from each page according to a *weighted round robin* scheduler using the rates as weights, and send new symbols from each page as it is scheduled. Unfortunately this method has two major problems. First, in a lossy network, some symbols may be dropped, requiring the client to issue retransmit requests for those symbols. In an asynchronous multicast setup, this approach does not scale. Second, and more importantly, multiple servers transmitting symbols from the *same* page need to co-ordinate their transmissions carefully to avoid duplicates. Variations in network conditions like bottleneck bandwidth, packet losses etc. compound the complexity of this approach.

A coding theory solution

Coding theory provides a simple and elegant solution to both problems. The idea is to encode a symbols using an *erasure-resilient* code and transmit the encoded symbols instead of the source symbols on the channel. A *maximum distance separable* (MDS) code encodes k source symbols into n encoded symbols in a way that any k of the encoded symbols suffice to recover the original k source symbols. In our setup, since each server transmits pages at different rates, a *Priority Encoded Transmission* (PET) is employed.

Given a message of m symbols and a priority function $\rho = \{\rho_1, \rho_2, \cdots, \rho_m\}$, a PET system produces n encoded symbols such that symbol s_i can be recovered from any ρ_i of the n encoded symbols. To apply PET to our protocol, assume that all servers have a common encoder $E_{\text{MDS}}(N, k)$, which, for $N \geq k$ and input of k source symbols, will emit N encoded symbols using an (N, k) MDS code [8]. Each server $S_i = \langle X_i, C_i \rangle$ upon receiving request R_i performs the following procedure:

First it encodes each page $P_j \in X_i$ with an $(N_j, |P_j|)$ MDS code, i.e., any distinct $|P_j|$ out of the N_j encoded symbols suffice to perfectly reconstruct page P_j. S_i then constructs n_i packets of a suitable length ℓ by choosing k_{ij} out of n_{ij} encoded symbols per page P_j per each packet p_{ik}, $k = 1$ to n_i without replacement, where* :

$$n_i = \frac{|C_i|}{\ell} \max\left\{\frac{|P_j|}{r_{ij}}\right\};\ N_j \geq n_{ij} = \frac{r_{ij} n_i \ell}{|C_i|};\ k_{ij} = \frac{\ell\, r_{ij}}{|C_i|}. \quad (4)$$

Each server S_i then sends the message $M_i = \{p_{ik} : k = 1 \text{ to } n_i\}$ on C_i.

Theorem 1

By transmitting M_i on C_i, the server guarantees rate r_{ij} for page P_j on C_i.

Proof: Without loss, at least n_i distinct packets will be received by any listening client and n_i packets suffice to recover all $P_j \in X_i$. A client listening for time t will get $\frac{t\,|C_i|}{\ell}$ distinct packets and hence $\frac{t\,|C_i|\,k_{ij}}{\ell}$ distinct symbols for page P_j. Using (4), this reduces to $t\, r_{ij}$. Thus $|P_j|$ distinct encoded symbols will be received in time $\frac{|P_j|}{r_{ij}}$. Due to the MDS code used, this suffices to recover P_j, and hence the rate of P_j is r_{ij} by definition. ∎

One problem remains: the client should not merely be able to reconstruct page P_j from $|P_j|$ distinct encoded symbols, but also from $|P_j|$ *encoded symbols received in parallel from multiple servers*. Thus not only should encoded symbols received from any server be distinct, but also encoded symbols *received across servers*. To achieve this, each server will use an $(N_j, |P_j|)$ MDS code, and truncate it by choosing n_{ij} unique encoded symbols. A lower bound on N_j is

$$N_j \geq \sum_{i=1}^{|S|} n_{ij} = \sum_{i=1}^{|S|} r_{ij} \max\left\{\frac{|P_j|}{r_{ij}}\right\} \quad (5)$$

Theorem 2

By each server S_i transmitting M_i on C_i, the client gets a rate $\sum_{i=1}^{|S|} r_{ij}$ for page P_j on C.

Proof: This is easily seen to be true as long as each server uses a truncated code and picks non-conflicting sets of symbols. This can be achieved by the client computing N_j and partitioning it into appropriately sized regions for each server. This information is then transmitted with the request. ∎

Feasible solutions

In principle, the abovementioned scheme guarantees duplicate-free transmissions from multiple servers. However, for large docments, the encoding and decoding complexity can be prohibitive.

In practice, we envision the use of codes with much more efficient encoding and decoding [9]. Specifically, we propose the use of the recently discovered *rateless codes* [3]. Rateless codes, instead of having a fixed encoding length, use *random* encoders to generate code symbols in a *stateless* manner, and are ideally suited for large-scale data distribution applications. The encoding process is extremely simple: to generate a code symbol, the encoder randomly picks a degree d from a carefully constructed degree distribution. It then picks d random source symbols, and computes the encoded symbol as a sum (usually bit-wise XOR) of these source symbols. A small disadvantage is that these codes are not MDS. To recover the original k source symbols, some $k(1+\epsilon),\ (1 >> \epsilon > 0)$ encoded symbols are needed. This simple structure leads to extremely fast encoding and decoding.

According to this alternate scheme, all the servers agree in advance to use a given rateless code. Just as in the straightforward scheme discussed above, the server uses the rate vector and a weighted round-robin scheduler to schedule page transmissions. The difference is that whenever a page is scheduled, instead of transmitting a source symbol from that page, the rateless encoder randomly generates a code symbol from that page and transmits that symbol over its channel. This scheme is illustrated in Fig. 1.

This approach solves all the problems associated with parallel transmission, without the prohibitive complexity of using regular MDS codes. For example, if the encoder uses a *Raptor code* [10], the encoder does $O(\log \frac{1}{\epsilon})$ work per symbol regardless of the number of pages or the rate vector. Decoding complexity is equally low, $O(|P_j| \log \frac{1}{\epsilon})$ for page P_j.

*Assume $\sum_{P_j \in X_i} r_{ij} = |C_i|$.

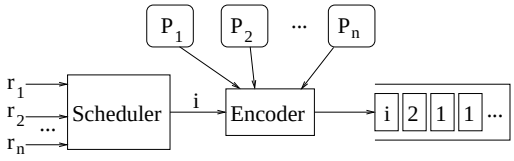


Fig. 1. Stateless Priority Encoding.

6. Applications

In this section, we discuss the use of these methods in bulk and streaming media distribution. In bulk data distribution, a single client downloads a data file in parallel from multiple servers. In streaming media distribution, a set of servers co-operate to transmit a video over a multicast session using a scalable transmission scheme as in [11].

Bulk distribution

In bulk data distribution, a receiver downloading from all the servers in parallel is interested in recovering the file in its entirety after receiving the minimal number of distinct symbols. Since the channel rate is $|C|$ symbols per unit time, ideally the client should recover D after time:

$$t_{min}(D,C) = \frac{\sum_{i=1}^{|D|} |P_i|}{|C|}$$

This can be achieved by choosing a schedule in which all pages finish downloading at time $t_{min}(D,C)$: Thus, for bulk data distribution the client picks a schedule ρ:

$$\rho(P_j) = \frac{|P_j|}{t_{min}(D,C)} = \frac{|C||P_j|}{\sum_{i=1}^{|D|} |P_i|} \quad j = 1, 2, \cdots, |D|. \tag{6}$$

Now consider when an optimal solution is not feasible: the schedule ρ cannot be met without exceeding server channel bandwidths for some servers.

Since the goal of the client is to download the *entire* document in the shortest time, the total download time may be taken to be the time taken for the *slowest* or the bottleneck page to download. If the aggregate rate or inflow to a particular page P_j is ρ_j, then it can be recovered in time $\frac{|P_j|}{\rho_j}$.

Hence the client sets up a cost function for each page as its additional download time:

$$C_{P_j}(\rho_j) = \frac{|P_j|}{\rho_j} - t_{min}(D,C) \tag{7}$$

Using this, the definition of ρ and the server descriptions, the client can determine a rate allocation that will minimize its total download time of D.

Streaming distribution

For bulk distribution, the goal of the client is merely to obtain the *entire* file as quickly as possible, and it suffices to be able to recover all symbols in the file at the end of the download process. In contrast, for streaming media applications, the file is consumed piecemeal: symbols earlier in the playout must be available before later symbols.

In this application, assume that asynchronous multicast is used. A set of servers co-operate to stream a movie over a shared multicast session. Peers join this session to retrieve the movie, playing it out after a short delay while concurrently downloading later parts of the movie.

For simplicity, assume D must be played out sequentially ordered from pages P_1 to $P_{|D|}$ at a uniform playout rate 1 symbol per unit time, so that the *playout interval* of page P_j is $|P_j|$. Further assume that after beginning the download process, each client will wait for time w ($w \geq 1$) before commencing playout. Thus page P_1 should be recoverable in time w, P_2 in time $w+|P_1|$,$\cdots$, and in general P_j in $w + \sum_{k=0}^{j-1} |P_k|$ for $j = 1, 2, \cdots, m$(Let $|P_0| = 0$).

In order to meet these requirements for continuous playout, the aggregate rates for each symbol should be assigned as:

$$\rho(P_j) = \frac{|P_j|}{w + \sum_{k=0}^{j-1} |P_k|}, \quad j = 1, 2, \cdots, |D|. \tag{8}$$

Note that w is a constant that is determined by the constraint $\sum_{j=1}^{|D|} \rho(P_j) \leq |C|$.

Using this schedule and the servers' descriptions, a rate allocation can be determined if it exists.

Consider when no feasible allocation exists. In this case we can formulate a mincost version of the problem. When server rates are constrained, some pages may take longer than $\rho(P_j)$ to download. For a client, this translates to a longer delay before playout; for continuous playout, the client has to wait for a time $w' > w$ such page P_j downloads in time $w' + \sum_{k=0}^{j-1} |P_k|$, and so the cost is just $w' - w$:

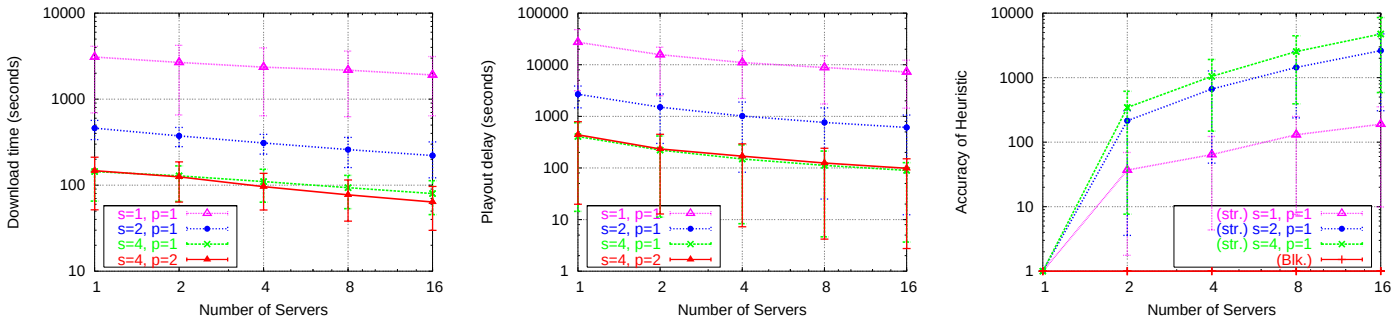
$$C_{P_j}(\rho_j) = \frac{|P_j|}{\rho_j} - \frac{|P_j|}{\rho(P_j)} = \frac{|P_j|}{\rho_j} - w - \sum_{k=0}^{j-1} |P_k| \tag{9}$$

7. Performance

The performance of the abovementioned methods is summarized in Figures 2(a) and 2(b). We used a trace of the *Gnutella* peer-to-peer network [12] collected during May 2001. These traces include information about bottleneck bandwidth of peers as well as the amount of data shared.

We used the traces in the following way: first we limited ourselves to those peers about which bottleneck bandwidth information as well as file-sharing information was available, to arrive at a list of about $50K$ nodes. The hypothetical document to download was a $1GB$ file divided into 100 pages of $10MB$ each. For the bulk download case, the goal was to download the entire document in the shortest time possible. For the streaming multicast scenario, the document was assumed to contain 8000 seconds of a $1mbps$ video, and the goal was to stream the document such that clients could start streaming it after the shortest delay possible. In both setups, the servers were randomly selected from the entire list, and σ_i and E were assigned according to the bottleneck bandwidths and amount of shared storage on each peer.

We used the following tuneable parameters: the server slack (s) is the ratio of servers randomly selected to the number of actual downloads initiated. (i.e., to select S servers, we randomly picked $s \times S$ servers and selected the S ones with most bandwidth. The page slack (p) is the ratio of pages of D (counting duplicates) present on these servers to the size of the document D. Using these parameters and trace values, we randomly assigned documents to



(a) Bulk (Trace-based) (b) Streaming (Trace-based) (c) Effi ciency of heuristic

Fig. 2. Performance of Parallel retrieval scheme: #servers vs. delay

servers. Each data point in the graph is an average over 100 runs; the error bars represent a 25% confidence interval.

Considering the high bandwidth variability of peers in these traces, the results are promising. We found that download delays could be shrunk by a factor of $3-7$ on an average as the number of servers went from 1 to 16. For a given number of servers, the performance increases dramatically with increasing server slack s, reflecting the high bandwidth variability of peers in the Gnutella network. More intelligent server selection would doubtless improve these preliminary results.

Fig. 2(c) shows the efficiency of our proposed heuristic as a ratio of delays using heuristic and optimal allocation (a value of 1 is ideal). We find that it performs admirably in *bulk distribution*, but fails miserably when used in *streaming distribution*. The reason is that for the scalable streaming scheme, the pages are transmitted at *widely* disparate rates; the first page is transmitted almost 100 times faster than the last one. Hence, a deficit in the transmitted rate of the last few frames translates to a much higher initial delay during playout. We are working on modifying the heuristic to take this into account. For our simulation setup, using Bloom filters was unrealistic. In an update to this paper, we will include performance numbers when using both Bloom filters and the rate allocation heuristic on a larger number of pages and servers.

8. Concluding remarks

In this paper, we proposed *priority encoded transmission* for parallel retrieval of structured documents from multiple servers according to a schedule. We also proposed a way to determine the optimal priorities to enable client to meet its schedule, and a way for servers to priority-encode their data in a stateless manner using recently invented efficient codes. Our current research focuses on the following:

Adaptive protocol: We have outlined a *static* solution that is optimal for a fixed set of parameters like server bandwidth etc. In practice, this should be *dynamic*. The biggest change is in the bottleneck bandwidths of servers as individual servers react to congestion by shrinking their congestion window (in the case of TCP) or by implementing a TCP-friendly algorithm (in the case of UDP), or by load-balancing their transmissions to multiple clients. In peer-to-peer systems the the set of servers is not static, but changes constantly as peers arrive and leave following the same process as clients.

We believe our protocol can be modified to be an *adaptive* one as follows: clients recompute priorities periodically based on measurements and send update messages back to the servers, and the latter adjust their encoding accordingly. In the case of asynchronous multicast, this could by done periodically by one of the servers in a randomized manner rather than rely on feedback from clients. The tradeoff of interest is the efficiency of transmissions versus overhead, the tuneable parameter being the frequency of updates.

Concurrent access: Another important extension is to consider multiple clients interested in retrieving different documents concurrently from an overlapping set of servers, thus being forced to share bottleneck links. The challenge is to devise a protocol that does not involve clients talking to each other; one way to do this is to let clients send their schedules directly to servers, which then co-ordinate with other servers to compute priorities in a globally "optimal" way.

Intelligent server selection: In our performance studies involving the server slack, we have chosen to select servers with the highest bottleneck bandwidth. More intelligent selection will involve better use of server descriptions to choose servers that will minimize download times.

Acknowledgment

We thank the participants of the *Snowtella* [12] project at the University of Washington for making their Gnutella traces available to us.

References

[1] R. Janakiraman, M. Waldvogel, and L. Xu, "Fuzzycast: Effi cient video-on-demand over multicast," in *Proceedings of Infocom 2002*, New York, NY, USA, May 2002.

[2] J. W. Byers, M. Luby, and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads," in *INFOCOM (1)*, 1999, pp. 275–283.

[3] M. Luby, "LT codes," in *Proc. FOCS 2002*, 2002.

[4] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan, "Priority encoding transmission," in *IEEE Symposium on Foundations of Computer Science*, 1994, pp. 604–612.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*. 2001, pp. 149–160, ACM Press.

[6] "CFSQP Software Page," http://www.aemdesign.com/FSQPframe.htm/.

[7] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol," in *Proceedings of the ACM SIGCOMM'98 conference*, Sept. 1998, pp. 254–265.

[8] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, Wiley-Interscience, 1998.

[9] M. G. Luby, M. Mitzenmacher, M. Amin Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation," Tech. Rep. TR-97-044, Berkeley, CA, 1997.

[10] M. A. Shokrollahi, "Raptor codes (Unpublished draft)," 2003.

[11] L. Xu, "Effi cient and Scalable On-Demand Data Streaming Using UEP Codes," in *Proceedings of ACM Multimedia 2001*, Ottowa, Canada, 2002.

[12] S. Saroiu and P. K. Gummadi and S. D. Gribble, "A Measurement Study of Peer-to-peer File Sharing Systems," in *Proceedings of MMCN 2002*, Apr. 2002.