Locality-Aware Content Distribution

Danny Bickson, Dahlia Malkhi and David Rabinowitz*

Abstract

We present the design and deployment of the Julia locality aware content distribution algorithm. Our novel contributions are locality aware node selection, forming a dynamically changing topology and division of the file into varying length chunks based on locality of the transfer. We present a large scale WAN deployment on over than 250 PlanetLab machines. We show that our technique can improve average download speed up by a factor of two compared with non locality aware solutions. Finally we compare our prototype's performance with that of BitTorrent, presently one of the most stable and efficient deployed content dissemination tools.

1 Introduction

Multicast overlays empower the network's leaf nodes in order to enhance the bare network functionality with the ability to disseminate content to multiple recipients. But operating above the network layer often results in inefficiently sending the same content over links multiple times, and unnecessarily transferring data to sites farther away than necessary. For example, recent measurements [7] show that most p2p searches could remain completely local, saving up to 80% of the search traffic.

This paper presents Julia, an efficient peer-based content dissemination (CD) tool with a focus on **locality awareness**. By peer-based dissemination we mean that peers that begin downloading content themselves become providers of the content to other peers, in order to parallelize the data dissemination. Various choices affect the design of a cooperative CD network, including how the data is split and spread in the network, how peers are selected for exchanging data, and others. Our work aims at optimizing three performance measures:

Work, the sum of transmitted bits, each multiplied by the distance over which it is transferred.

Time, the overall elapsed duration from initiation and until all recipients obtain the content.

Fair-Sharing, the upload to download ratio.

Motivation. The application we aim for is a CD tool for on-the-fly simultaneous download of a sizable content from a single source. Such bursts of 'flash crowds' are frequently observed when a new file is first uploaded onto file sharing networks like KaZaA, or when a new software distribution appears on the network. Cooperative solutions are appealing either for the end clients, or in order to form an intermediate group of servers (like the Akamai network) that serve replicas of the content and thereby increase the available upload throughput.

A characteristic of our aimed application is that its membership gets formed on-the-fly, and so the overlay must form quickly and spontaneously. Several existing CD networks build on a long-lasting structured topologies, e.g., SplitStream [2] over the Pastry overlay [9], and Bayeux [13] over Tapestry [12]. The drawback of these schemes is the overheads of membership maintenance and topology repair. Our solution is unstructured, inherently fault tolerant and adaptive to changes in the network.

Driven by the scale which we aim for, we put our strongest emphasis on locality-awareness, i.e., on minimizing the Work incurred on the network. Locality considerations exists in various previous approaches. Application-level multicast overlays, e.g., Narada [3] and TMesh [10], build a spanning tree rooted at the source, that heuristically approximates the underlying network structure. Other CD schemes derive their locality-awareness from an underlying routing overlay, e.g., SplitStream [2] and Bayeux [13]. In SRM [6] missing packets recovery is done

*School of Computer Science and Engineering, The Hebrew University of Jerusalem. Email: {daniel51,dalia,dar}@cs.huji.ac.il

locally. BitTorrent [5] uses another approach for optimizing the download time by constantly selecting a small subset of links with the maximal bandwidth out of a larger group of links. Julia's approach to locality awareness is different from all of the above schemes. It is designed to minimize the overall Work, and derives from the formal basis laid in [1]. The Julia scheme considers the overall Work both in its selection of links and in scheduling data transfers. As a result, Julia avoids to a large extent sending content repeatedly over long-haul links. (This is defined as link *stress* in [3].) Locality-awareness promotes Julia's scalability and makes it attractive for use in practice.

Another important design goal is to promote simultaneity, and at the same time, prevent free riding by nodes which participate in the download but do not contribute their upload resources. Using symmetry of connections, where a pair of nodes meet and mutually exchange one part of the file, Julia has a balanced exchange mechanism. Load balancing leads both to formal decrease in overall delivery time, and to good Fair-Sharing in practice.

Our Approach. Our contribution and the lessons we learned in building Julia can be summarized as follows. First, a key feature of Julia is symmetry. Symmetry in Julia prevents inactive waiting periods, and lets all participants be active throughout the protocol. Another advantage of symmetry is that a "tit-for-tat" policy can be easily implemented since the nodes mutually exchange information upon each encounter. Our deployed system indeed exhibits Fair Sharing close to one, as detailed in Section 4.

Second, Julia employs a sophisticated, adaptive policy that determines the amount of data exchanged between pairs of clients. It sets the amount of data in an exchange according to the progress of the protocol, and according to the distance of the exchange-partners: The protocol moves from small amounts of data over long-haul links in the beginning, to increasingly larger amounts of data over short links towards the end. In this way, while two close-by partners exchange a lot of data, two remote partners exchange smaller amounts, and simultaneity is again maintained.

Third, Julia builds a dynamic locality-aware mesh of links among downloading partners. Nodes form connections based on their download progress: at the start of the download the links are formed at random. As the download progresses, more information is gathered about the network, and increasingly closer links are chosen. Thus, most of the file is transfered as locally as possible.

Our solution has no structured topology, since the topology is constantly changing based on the downloading node's needs. Furthermore, unlike structured solutions, there is no group management overhead since the groups are formed dynamically. Fault tolerance is inherent, since disconnects do not disrupt any predefined topology. As a consequence, the protocol overhead is minimized.

Today, an initial version of Julia is operational and has been deployed over the PlanetLab distributed testbed [4] as well as on our LAN. At the time of this writing, we are employing over 250 Planet-lab nodes to rapidly distribute files of more than 100 MB in as many as 500 smaller chunks. Our deployment approach of Julia is highly flexible and fault tolerant. Measurement shows that using locality awareness can reduce download speed almost down to half the time of random node selections. Furthermore, we compare our system to a state-of-the-art fully optimized system like BitTorrent [5], and show that we achieve comparable performance.

2 The Julia Protocol

The Julia system borrows its principles from the Julia protocol introduced in [1]. Julia is a content delivery algorithm which is locality-aware. It is designed to simultaneously minimize download Time, reduce Work, and balance the load between participating nodes.

Let us call the content that we wish to deliver Foo from now on, and denote its size by $|Foo|$. Julia divides the file Foo between the set of n participants, and then utilizes a mesh of links in order to simultaneously exchange pieces of Foo among them.[1] Unlike previous protocols, the Julia design emphasizes locality awareness during the exchange phase, in order to minimize Work. Moreover, Julia is not tree-

[1] A straight-forward extension is to employ an erasure code to transform Foo into n parts, such that $n - t$ parts, for some parameter t, suffice to reconstruct Foo.

based, as most previous protocols, but rather utilizes simultaneous symmetric exchange to decrease Time.

The protocol is based on the following three principles:

1. Each node maintains links to nodes at varying distances, from closest to farthest.

2. Nodes generally choose exchange partners at progressively lower distances, starting from far-away nodes and down to the near-by nodes.

3. The amount of data exchanged between nodes is in reverse proportion to their distance.

There are several benefits from following these principles. Far apart nodes, which are likely to engage in the exchange early on in the protocol, exchange small pieces of data. This progressive behavior contributes to Work, since data travels only once to remote locations, and is then exchanged locally. To the contrary, close-by pairs, acting toward the end of the protocol, exchange large amounts of data. This guarantees fast completetion of the protocol.

Additionally, the reverse distance-size principles promotes parallelism overall, as (roughly) the same amount of time is dedicated to exchange large sizes of data over local links as small data sizes over long-haul links.

2.1 A Formal View

A formal view of the Julia protocol is as follows.

Geometry: Real network distances determine the links of every node in Julia. Each node has $\log(n)$ outgoing links. For a node u, enumerate all other nodes $u_1, u_2, ..., u_{n-1}$ according to their distance from u, from closest to farthest. Node u's level-i out-link is $u_{n/2^i}$, the $\frac{n}{2^i}$'th closest node to u. (Thus, the level-$\log(n)$ link is the closest neighbor of u.)

A useful intuition on the topology of Julia is that of a locality-aware hypercube, as depicted in Figure 1. The two main faces of the hypercube (front and back) are the most geographically distant from one another. By way of an example, we may have one face contain all the nodes in the north American continent, and the other face has European nodes. Level-1 links of the nodes on either face 'cross the atlantic' to their half-network neighbors away.

Within each face, nodes are split geographically along the vertical split (so left nodes are on the west coast, and right nodes on the east coast). Level-2 links cross between the left and the right sides of each face. And so on.

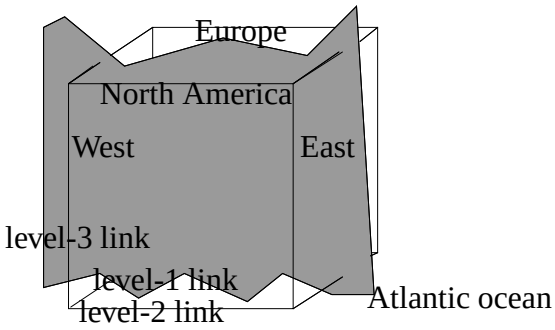

Figure 1: Julia's locality-aware topology.

The next question is of course how to disseminate Foo along this "hypercube" in a Time and Work-efficient manner.

Protocol: As a first step, the source of Foo sends pieces of Foo to all participants. Each piece has size $\frac{|Foo|}{n}$. The remainder of the protocol consists of $\log(n)$ logical rounds. In round i, each node exchanges all the pieces it holds with its level-i link partner. In this way, round 1 consists of exchanges of single pieces, each of size $|Foo|/n$, sent over half the network away. And generally, round i consists of exchanges of 2^i pieces, totaling in size $|Foo|2^i/n$, over a distance of $1/2^i$. In the last round, nodes exchange with their closest neighbors data of size $|Foo|/2$.

Protocol Properties: In each round of the Julia protocol, every pair of nodes that exchange data holds disjoint pieces of Foo. Thus, the pair of nodes exchange **all** the information they accumulated up to this round. As a result, the protocol enjoys both simplicity and complete symmetry. In particular, all nodes experience the same load, and no node can be a free rider. In summary, the **Fair-Sharing** measure of Julia is exactly 1.

Julia locality awareness is manifested in the fact that the size of an exchange is reversely related to its distance. A precise analysis of Julia's Work along a grid of $D \times D$ shows that Julia's **Work** is $|Foo|\frac{D}{2}(\log(n)+1)$. All previously known schemes, including [2, 8, 13] have Work $O(|Foo|Dn^{1-\frac{1}{\log(n)}})$ in the same topology settings.

For **Time**, we note that all the nodes are busy downloading all the time, and no nodes are idle waiting for internal bottlenecks to complete. The time for the download of the full file is $2|Foo|$, which is very near to optimal. Time optimality in a uniform-bandwidth model is almost achieved in [11], where translated to our notation the Time is $|Foo|\left(1+\frac{log(n)}{n}\right)$. Likewise in SplitStream [2], time is $|Foo|+c\log(n)$, where c is a parameter that depends on tree degree and the number of chunks Foo is split into.

3 The Deployment

We have implemented a content distribution client following the Julia algorithm, and tested it in both LAN and WAN environments. The client software is written using C++ and consists of about 15,000 lines of code. In each file transfer there is one *source* node or seed, who possesses the file at the outset. The file itself is divided into pieces called chunks. Source nodes decide which chunks to give out to requesting nodes. Client nodes connect to the source, obtain some chunks from it, and learn about other active peers that are downloading the same file. At that point, clients connect with their peers and start exchanging chunks with them. Once the client has finished, it becomes a source node.

Protocol messages: The implementation makes use of a simple protocol composed of only five messages: FILE_INFO_REQUEST, FILE_INFO_REPLY, CHUNK_REQUEST, CHUNK_REPLY and an error message. A new client contacts any existing participant, and sends a FILE_INFO_REQUEST message. The response is a FILE_INFO_REPLY message, containing information about the file, such as its size, number of chunks, CRC of chunks, and so on. The FILE_INFO_REPLY message also contains a fixed size list of other connected nodes, that are currently participating in the download. If the responding participant is not too loaded, it also sends a CHUNK_REPLY with one chunk of data. After it obtains this initial list of participants, the new client proceeds to contact other participants with CHUNK_REQUEST messages. The response to CHUNK_REQUEST is a a CHUNK_REPLY message, containing a list of other participants, along with chunk data.

Geometry: In our system, links are chosen based on distance estimation, whereby distance we mean a weighted value composed of observed bandwidth and latency. We classify node-distances into log-scale classes, corresponding to the logarithmic link levels of Julia: The farthest half, next a quarter, next an eighth, and so on. In our experiments, we use eight levels, corresponding roughly to $\log(n)$ where $n = 250$ is the number of PlanetLab machines in our deployment. Within each distance-class, we randomly select a target, and maintain information about several potential replacements. As messages keep flowing in the network, nodes may learn of more suitable target nodes and replace certain links, or may learn of link failures and reassign them.

The estimation of distances among nodes is done gradually, using bandwidth and latency information gathered from the exchange of chunks itself. At the start of the download, a new node obtains information about other downloading nodes, but does not have distance information about them. As the download progresses, bandwidth and latency data recorded from the chunks transfer is used for categorizing the nodes into levels.

Thus, no communication overhead is incurred on network probing. Furthermore, the overhead does not depended in the number of participating nodes since the routing updates have fixed size. Our experimental measurments indicate that Julia's protocol overhead is between 0.1% - 0.3% of the transmitted data. Under similar conditions, Bullet has an overhead of about 6% [8], SplitStream control overhead is 0.17% (not including the topology construction), and Narada has an overhead of up to 2% depending on the group size [3]. In summary, this deployment strategy has sufficient flexibility and redundancy so that maintaining the network in scalable settings is manageable, and coping with churn is simple.

Protocol: The deployment of the Julia protocol faces several interesting challenges in practice.

First, distance information is not available to participating nodes initially, and furthermore, it is gradually collected and keeps changing. To address this, at the start of the download, a node selects other nodes for exchange at random. As the download progresses, distance information is collected about more and more nodes, and they are chategorized into log-scale levels. Using this level information, closer

and closer nodes are contacted. Towards the middle of the download, the probability of contacting a far-away node decreases, and as the protocol reaches its final stages, only nodes in the close vicinity of the node are contacted.

Second, the deployed system cannot follow the rigid chunk-selection schedule of the Julia protocol of Section 2 above precisely. Furthermore, good chunk selection strategies, e.g., based on how rare a chunk is, turned out to be crucial for achieving good performance. Nevertheless, the general principle of exponentially increasing batches of chunks is followed in the implementation. When multiple chunks are grouped into one exchange, they are batched together in one TCP/IP stream to improve performance.

In order to decide which chunks to exchange among nodes, exchange partners transfer between them bitmaps representing the nodes' available chunks. Each chunk can be in one of three states: HAVE, BEGUN or MISS. The receiving node is responsible for deciding which chunks to give to the requesting node and which chunks to get from it. In addition to the bitmap of the current nodes it communicates with, each receiver also keeps the bitmap it has heard in the past from every node it communicated with. This is used to obtain a rough estimate of the frequency of chunks in the network. The receiving node decides which chunks to give and to get based on their frequency in the network, where rarest chunks are given first (the Bittorrent system has a similar policy [5]).

Third, as we gained experience with the system, we made several important optimizations. One has to do with allowing simultaneous exchanges. Originally in Julia, nodes perform the exchanges sequentially, starting with far-away nodes and gradullay moving closer. Since the far nodes usually have lower bandwidth and waiting for the first chunks to arrive takes a while, our system supports simultaneous exchange across levels all at once. This is an important optimization, which boost performance cosiderably. Additionally, following Bittorrent [5], we implemented an "endgame mode" where the last chunks are requested from all known nodes in order to speed up the completion of the download. Practice shows that not much bandwidth is wasted since the chunks arrive from the fastest node first, and the other requests are canceled.

4 Performance Results

Our experiments were done on both LAN and WAN environments, but due to space constraints, we report only about the WAN experiments here. We used the PlanetLab testbed for WAN experiments, using over 250 machines located at about 120 sites around the world. The PlaneLab machines vary in configuration and bandwidth. In all experiments, we used file sizes of either 130MB or 30MB, and a chunk size of 500KB. All experiments where repeated 5-10 times and the result was averaged.

We conducted several experiments. In order to give some meaning to the performance results, we give as reference in all of the experiments the performance of BitTorrent in the same settings. (Appendix A provides more details on the structural differences between the two systems.)

First, in order to evaluate the effect of locality-awareness in the Julia exchange-partner selection strategy, we compared it against random exchange-partner selection. The results of downloading a 30 Mb file are depicted in Figure 4. The graph demonstrates that the average time for downloading the file was reduced to about half using Julia's locality aware node selection, compared with random node selection.

Second, we evaluated the Fair-Sharing of Julia. The Fair Sharing values experienced in Julia are summarized in Figure 3. About 93% of the nodes have a Fair Sharing ≤ 1.6. Only 1% of the nodes in Julia have Fair-Sharing of 2 or above. Once again, we provide the Fair-Sharing values of BitTorrent for reference, yielding about 72% of the nodes with Fair Sharing ≤ 1.6, and about 14% of the nodes with 2 and over.

Third, we investigated the effect of several chunk selection strategies. As shown in figure 4, exchanging rarest chunks first performs better better than random chunk selection, and outperforms a mixed policy of switching between rare and random chunk selection arbitrarity.

In conclusion, due to its high flexibility and optimized link utilization, we expect Julia to become an attractive substitute for large scale content delivery.

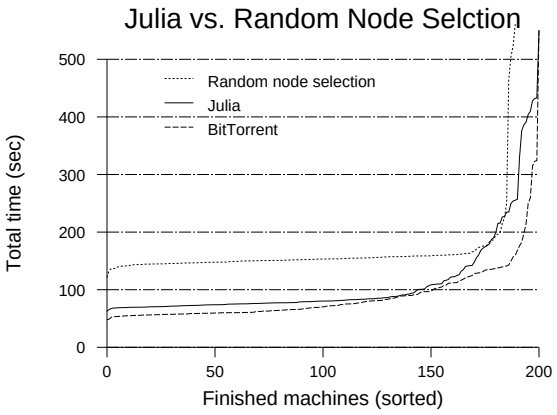

Figure 2: *Locality-aware vrs. random node selection. Lower dashed line is BitTorrent performance for reference.*

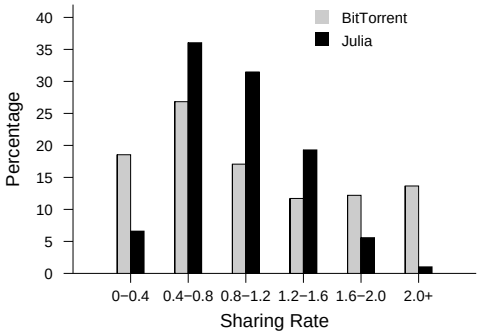

Figure 3: *Fair-Sharing (upload/download ratio).*

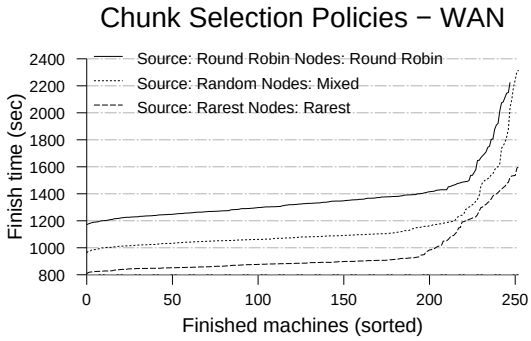

Figure 4: *Different chunks selection strategies in the WAN. File size is 130Mb. Node selection is random.*

References

[1] D. Bickson, D. Malkhi, and D. Rabinowitz. Efficient large scale content distribution. In *The 6th Workshop on Distributed Data and Structures (WDAS'2004)*, July 2004.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP'03*, October 2003.

[3] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *In Proceedings of ACM SIGMETRICS, Santa Clara, CA, pp 1-12*, June 2000.

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broadcoverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[5] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of P2P Economics Workshop*, 2003.

[6] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *IEEE/ACM Transactions on Networking, December 1997, Volume 5, Number 6, pp. 784-803*.

[7] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329. ACM Press, 2003.

[8] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SOSP*, October 2003.

[9] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[10] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *In Proceedings of Fourth International Workshop on Networked Group Communication (NGC)*, Obtober 2002.

[11] X. Yang and G. de Veciana. Service capacity of peer to peer networks. In *INFOCOMM 2004*, 2004.

[12] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[13] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.

A Comparison with BitTorrent

Julia aims at becoming a practical, scalable wide-area content distribution facility. We therefore report on its performance side by side with that of the widely deployed BitTorrent system [5]. In this section we outline the structural differences between Julia and the Bittorrent systems. A detailed description of BitTorrent is found in [5].

BitTorrent has a special nodes acting as a tracker for the network, and a different node as a source for each individual file. In Julia, the tracker and seed are in one node called the *source* node, and that node may be replicated for fault tolerance. When a source node is not heavily loaded, it acts as a seed for the file and provides data chunks to clients. When it becomes loaded, it acts only as a tracker, giving clients a list of other connected nodes.

BitTorrent maintains a rather static mesh of links for each download, with a default of twenty connections per node. Some of the links become idle as download progresses. *Choking* is employed for supressing lower bandwidth links and preventing them from becoming too congested. In contrast, the flow control and linking mechanisms in Julia are more dynamic, links being formed and torn down as exchanges get formed and complete. We believe this is suitable for very dynamic and fault prone environments. Rather than choking low bandwidth links, Julia boosts performance by choosing closer nodes as the download progresses. This aims at optimizing the network utilization.

BitTorrent fragments files into chunks of sizes that are powers of 2, and transfers only contiguous chunks. In Julia, any collection of fix-size chunks may be exchanged, thus avoiding fragmentation issues.

Table 1 provides a snapshot of the efficiency and resource utilization of the two systems in a download of a 30Mb file. The performance tests use the same node as BitTorrent's seed and as Julia's source node, and the same set of client machines. BitTorrent's chunk size is by default 256Kb, and for the Julia implementation we used a 500Kb chunk size. As the table indicates, on both systems, about 75% of the machines finished the download in the time frame of 95 seconds. (The remaining 10% of machines were very slow, some of them using satellite or ADSL links.) These results demonstrate that Julia's performance is comparable to cutting edge content distribution systems currently in deployment. At the same time, note that Julia source node sends the file data on average 1.9 times over the network, whereas BitTorrent sends it 2.66 times. Testing the actual Work performed by these systems in reality is not easy, and is currently under investigation.

	Julia	BitTorrent
Average download time (sec)	95	77
75% finish time (sec)	92	90.7
Average memory usage (Mb)	8Mb	4.5Mb
Source/seed fair sharing	1.9	2.66
Average node fair sharing	0.94	1.07
Number of connections	6	20 (default)

Table 1: *Comparison of Julia vs. BitTorrent characteristics. File size is 30Mb.*