

The Julia Content Distribution Network

Danny Bickson* and Dahlia Malkhi**

Abstract—Peer-to-peer content distribution networks are currently being used widely, drawing upon a large fraction of the Internet bandwidth. Unfortunately, these applications are not designed to be network-friendly. They optimize download time by using all available bandwidth. As a result, long haul bottleneck links are becoming congested and the load on the network is not well balanced.

In this paper, we introduce the Julia content distribution network. The innovation of Julia is in its reduction of the overall communication cost, which in turn improves network load balance and reduces the usage of long haul links. Compared with the state-of-the-art BitTorrent content distribution network, we find that while Julia achieves slightly slower average finishing times relative to BitTorrent, Julia nevertheless reduces the total communication cost in the network by approximately 33%. Furthermore, the Julia protocol achieves a better load balancing of the network resources, especially over trans-Atlantic links.

We evaluated the Julia protocol using real WAN deployment and by extensive simulation. The WAN experimentation was carried over the PlanetLab wide area testbed using over 250 machines. Simulations were performed using the the GT-ITM topology generator with 1200 nodes. A surprisingly good match was exhibited between the two evaluation methods (itself an interesting result), an encouraging indication of the ability of our simulation to predict scaling behavior.

I. INTRODUCTION

Peer-to-peer content distribution networks are becoming widely utilized in today's Internet. The popular file sharing networks—e.g. eMule, BitTorrent and KaZaA—have millions of online users. Current research shows that a large fraction of the Internet bandwidth is consumed by these applications [10]. Most existing solutions optimize download time while ignoring network cost, and put network load balance only as a secondary goal. As these networks become more popular, they consume increasing amounts of network bandwidth and choke the Internet. Eventually, their own performance deteriorates as a result of their success.

The approach we put forth in this paper takes network cost and balance into account from the outset. As in

* School of Computer Science and Engineering, The Hebrew University of Jerusalem. daniel51@cs.huji.ac.il.

** Microsoft Research Silicon Valley and School of Computer Science and Engineering, The Hebrew University of Jerusalem. dalia@microsoft.com.

The work presented in this paper was supported (in part) by the European project Evergrow No 001935.

most existing solutions, the fundamental structure of our content delivery algorithm relies upon an origin node (or nodes) which stores a full copy of the content and which serves *pieces* of the content to a set of downloading clients.

The clients subsequently collaborate to exchange pieces among themselves. The novelty in our approach is that communication partners as well as the pieces exchanged with them, are chosen with the aim of reducing overall network usage, while at the same time, achieving fast download time. All of this is accomplished while maintaining tit-for-tat load sharing among participating nodes, which is crucial for incentivizing client participation.

Our consideration of the total network costs for content dissemination adopts similar goals to those considered by Demers et al. [7] in the context of gossip algorithms. Their spatial distribution algorithm is aimed to reduce the communication costs of disseminating a file in a network. Their basic idea is to prefer closer nodes: this is done by setting the cumulative probability of contacting a node to diminish exponentially with distance. Simulation results show that this technique significantly reduces the communication work, especially over long communication links. Our distance-aware node selection strategy closely follows this spatial distribution algorithm, with two important distinctions. First, our node selection policy changes over time, and adapts to the progress of the algorithm. Second, we vary the amount of data that is exchanged between nodes, and adapt it to the progress of the download.

The Julia algorithm has its roots in an earlier algorithm proposed by us in [2] for disseminating content over a structured hypercube topology. In this work, we propose a new algorithm to handle arbitrary network topologies, provide simulation results to confirm the design goals, and highlight real WAN deployment results over the PlanetLab [5] testbed.

Encouraging results are exhibited using two complementary evaluation methods, extensive simulations and a thorough PlanetLab testing over WAN. The two are compared against the BitTorrent [6] network under similar settings. Both simulation results and real planetary scale testing confirm our design goals: the network load balance over nodes and links shows improvement, while

at the same time the communication cost is significantly reduced. However, our system pays little in terms of running time.

The rest of this paper is structured as follows: In section II, we present the Julia algorithm. Next, we discuss protocol implementation in section III. In section IV, we report experimental results from both simulations and the PlanetLab test-bed. Finally, in section V, we present an improvement to the Julia algorithm and discuss its feasibility.

II. THE JULIA ALGORITHM

In this section, we introduce the Julia content distribution algorithm, which aims for the efficient transfer of large files (at least tens of megabytes).

One of the first design decisions we had to make in Julia is whether to use some predefined structured communication overlay. We favored an unstructured, constantly changing mesh, which is resilient against failures and requires no maintenance. In terms of data dissemination, having an unstructured mesh means that any pair of nodes can choose to exchange information. In the remainder of this section, we discuss the strategy for exchanging file pieces among nodes.

The main emphasis in the design of the Julia protocol is to reduce the overall communication cost of a file download, and to incur a balanced load on the network, without significantly impairing download completion time significantly. These design goals led to a probabilistic algorithm that determines which node to contact at every step. As in the spatial gossip algorithm [7], we prefer downloading from closer nodes whenever possible. However, the Julia node selection strategy is unique in that it adapts itself to the progression of the download. This adaptation is done roughly as follows. At the start of the download, the nodes do not have any information regarding the other nodes' bandwidths and latencies. Hence, each node will select nodes for pieces-exchange at random. As the download progresses, the nodes gossip and gather statistics about the network conditions. This knowledge is then used in order to contact progressively closer nodes.

In addition to the distance, we also vary the amount of data that is exchanged between interacting nodes: at the beginning of the download, we send a small number of pieces across each connection. As the download progresses — and as the quality of connections we utilize improves, we gradually increase the amount of data sent.

More formally, we have a file for download F , of size $|F| = k$ parts. Let x denote the number of pieces a node holds. The *progress* of a node is defined as $\frac{x}{k}$. The *distance* between nodes refers to the communication cost

between them (the concrete parameters that determine the distance are an implementation matter; more on this in Section III. We use D_i to denote the maximal distance from node i to any other node.

The algorithm: Each node performs the selection of other nodes based on the following algorithm. Intuitively, we select nodes with an exponentially diminishing distance relative to the download progress.

Formally, we define $Q_i(d)$ as the set of nodes at a distance d or less from a node i . $Q_i(d)$ is known to i approximation only based upon the statistics gossiped during the download. Let node i have progress x/k . At each step of the algorithm, node i sets d to a value that reflects the download progress, using the exponential distribution formula $d = d(x/k) = D_i e^{-x/k}$. Node i then selects its next exchange partner uniformly at random from among all nodes in $Q_i(d)$, i.e., a node at distance up to d .

In this way, $x/k = 0$, at the start of the download, so that the initial selection is made from the entire universe of nodes $Q_i(D_i)$. When the download progress is about a halfway through, nodes from the closer group $Q_i(D_i e^{-1/2})$ are chosen. And so on, until close to the completion of the download, only very close nodes are selected.

III. THE IMPLEMENTATION

We implemented a content distribution client in C++ based on the Julia algorithm. The client is implemented using a single thread server queue. The implementation code consists of approximately 15,000 lines of code, and uses TCP for the transport layer. To improve performance, the client maintains several (we used six) parallel connections. That is because larger number of TCP parallel connections result in poor download performance¹. The decision of which node to contact next is made using the Julia algorithm.

One of the questions we had to answer when applying the Julia algorithm was how to calculate network distances. Different applications might have different views about distance. For example, streaming applications generally regard the communication latency as the distance, whereas file sharing applications usually consider the bandwidth as the main parameter to optimize. Other possible metrics include the number of hops or commonality of DNS suffixes. Additionally, local area links are cheaper to use than metropolitan links; metropolitan are cheaper than national links; and so on.

¹The same is done in the BitTorrent system where the actual downloading set of neighbors (out of the total neighbors set) is of size 4-5 [6]

Our goal of reducing the communication cost dictates that we must use a combination of these parameters. We took a similar approach for the Tulip routing overlay [1], and achieved a near optimal performance of routing. In Julia, we measure distance with a combination of bandwidth and latency. Note that latency is a good estimate of a link’s physical length and, therefore, of its cost. However, we do not want to take only latency into account because this might interfere with the selection of high-bandwidth links.

Estimating distances in practice is another pragmatic challenge. The Julia client starts the data dissemination process with no knowledge of network conditions. Since we decided not to spend any extraneous bandwidth on active network probing, network conditions are discovered by passively monitoring the transfer rate of uploaded and downloaded file pieces. As information about network links is gathered, the client can apply the Julia algorithm to decide which neighbors to communicate with out of the known nodes. Note that this gradual process fits well with the Julia protocol, since early node selection in Julia inherently has great flexibility.

One important issue left out of the discussion so far is the strategy for selecting file pieces to send and receive. A Julia client maintains a bitmap of the pieces it has obtained so far. This bitmap is used in an exchange in order to ensure that only missing pieces are transmitted. Additionally, the client locally records the bitmaps that other clients have offered in previous rounds. This information is used for estimating the availability of file pieces throughout the network. As shown in [9], local estimation of file piece frequencies is a good approximation for global knowledge of the real frequencies.

Among those pieces missed by an exchange partner, our strategy is to send the rarest piece first. We adopted this strategy as a result of extensive experimentation with several selection policies [3].

IV. EXPERIMENTS

A. The Simulation Method

The following are the performance measures we use in this paper: The download finishing *time* of a node is the time from the start of the download until the node has completely downloaded all file pieces. *Fair sharing* is the ratio between the number of file pieces the node forwards to the number of file pieces it receives. (In [4] this is called node stress.) Communication *work* is the product of file pieces traveled on a link and the link cost, summed over all the links. (In [4] this is called resource usage.)

Our simulation is done using a synchronous discrete event simulation we wrote, consisting of 3,000 lines of Java code. For the topology, we used the Georgia Tech topology generator (GT-ITM) [8] to create a transit-stub topology. We assigned stub-stub and stub-transit links bandwidth of 5 pieces per round, and transit-transit links bandwidth of 15 pieces per round. We used the link latencies, as created by the GT-ITM, to determine the link cost. The routing over the physical layer was done using Floyd all-pairs-shortest-path algorithm.

Out of the total of 600 physical nodes, we selected 200 random nodes to participate in the content distribution network. For each simulation, one source node was selected at random out of the 200 participating nodes. Each simulation was repeated at least 10 times and the results were averaged.

B. The PlanetLab Testing Method

Our PlanetLab test is done with a single source node storing the file in full, and about 250 nodes downloading simultaneously. The source node is used both for tracking other clients, and for retrieving pieces. Under a normal load, the source node provides a client that contacts it one data piece, the rarest, as well as a list of other nodes that previously connected to it. When the source node becomes overloaded, it stops serving pieces and provides only the list of nodes. After contacting the source node, clients exchange file pieces among themselves. We used file sizes of 30, 60 and 130Mb in our tests. Part size was set to 1/2Mb.

C. Preliminary Discussion of Results

It is enlightening to compare the simulation results to the real WAN experiments. The simulation environment is only a simplified approximation of a real system: nodes operate in synchronous rounds; the transmission of a piece is never disrupted; and all pieces sent in a round arrive before the start of the next round. Additionally, there are only two bandwidth categories, slow and fast. Reality is naturally more complex: No synchronization; heterogeneous machine capacities and diverse links; and there are node and network link failures, packet losses, congestion and unexpected delays.

Nevertheless, as we shall see below, a surprisingly good match is exhibited in our simulations of the PlanetLab settings. This is encouraging, as it suggests good prediction power for the simulation. The results below also indicate places where the simulation method may be improved for better accuracy.

D. Fair Sharing

Figure 1 provides a comparison of fair sharing in Julia and BitTorrent using both simulation and by deployment

over PlanetLab. Overall, we observe a remarkably close match between the simulation results and the WAN measurements. This can be explained by the fact that fair sharing is an algorithmic property of the protocol, and does not relate directly to bandwidth, or to the heterogeneity of the nodes.

The average fair sharing of both algorithms is a little less than one, which means that, on average, the network is load balanced. However, we can see that the Julia protocol provides a better load balancing of nodes, both for the simulation results and for PlanetLab. Surprisingly, WAN results show that, in practice, BitTorrent has a slightly higher fair sharing ratio than predicted. In contrast, the Julia client has a better fair sharing ratio than predicted (that is, closer to 1). We note that fair sharing is of immense importance for Peer-to-peer networks since it provides incentive to use the network.

E. Finishing Time

Figure 2 shows the completion times of our experiments. Here, the simulation and the PlanetLab results exhibit a slightly lower degree of matching than the Fair Sharing results above.

We speculate that the differences between the finishing times predicted by simulation and ones experienced through the PlanetLab tests are because the transit-stub model we use does not capture all of the PlanetLab network properties. For example, some machines in Brazil and Russia were behind lousy links, which made TCP perform poorly due to the slow start mechanism. Some of the machines are connected using ADSL, with asymmetric bandwidth properties, and had a narrow upload capability. Other machines were heavily loaded

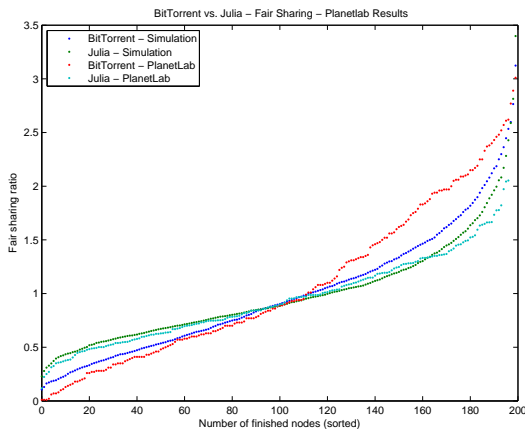


Fig. 1. Comparison between simulation and WAN results of fair sharing (node stress). Fair sharing of 1 means that the node uploaded the same number of file pieces it downloaded from the network.

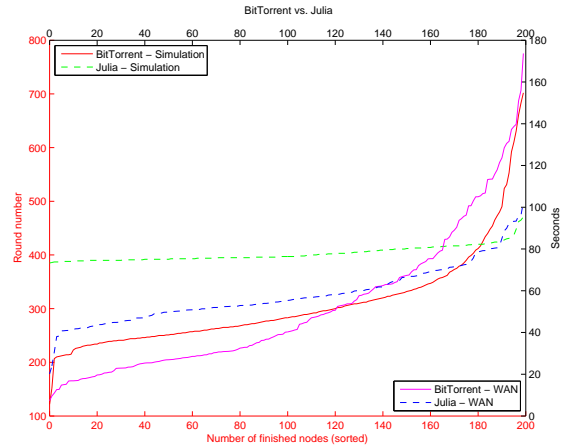


Fig. 2. Finishing times for 200 nodes using simulation vs. PlanetLab results. The left Y-axis represent simulation rounds and the right Y-axis represent time in seconds. Note that the 50 slowest PlanetLab nodes were not shown in the graph because of their exponentially increasing finishing times, probably because of very slow or congested machines.

and performed poorly. Our simulation did not capture those network properties well.

F. Communication Cost

Our evaluation of the total communication cost is done only by simulation, since on PlanetLab, evaluating the costs incurred in practice is a challenging problem, mainly because there is no unified distance measurement. In our simulation, we used the link latencies as created by the transit-stub model for link costs.

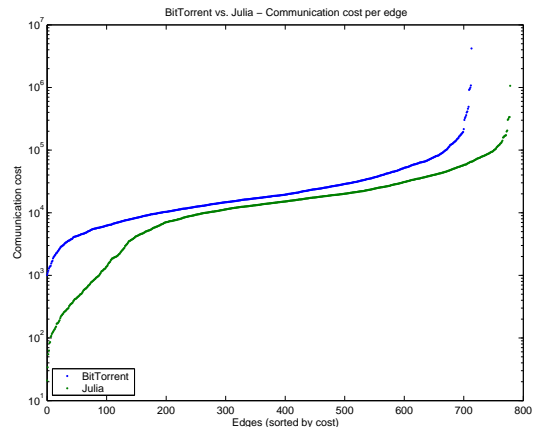


Fig. 3. Total communication cost per edge in simulation. The average communication cost of transferring a file to a node in the Julia algorithm is reduced by 33% relative to the BitTorrent algorithm.

Figure 3 shows simulation results of the communication costs per network link. The y-axis has a logarithmic

cost scale. The x-axis presents the links ordered by their communication cost. Links with cost zero were removed from the graph. We can clearly see the advantage of using Julia, resulting in a reduced network load. Simulation shows that the average communication cost of transferring the full file into each node is lowered by 33% relative to the BitTorrent algorithm.

We conducted an additional simulation, whose goal was to evaluate the load incurred on a costly trans-Atlantic link. To this end, we took two transit-stub networks of 600 nodes and connected their backbone using one link. The links in each network had bandwidth 5 pieces per round for transit-stub, and 15 pieces of round for stub-stub links. The trans-Atlantic link was assigned a bandwidth of 150 pieces per round. Two hundred nodes were selected at random to perform the overlay out of the total 1,200 physical nodes. We ran both the Julia and the BitTorrent algorithms to compare the number of file pieces traveled on the trans-Atlantic bottleneck link. As expected, this link was used in BitTorrent to transfer as much as four times the number of pieces relative to Julia. We conclude that the Julia algorithm has a potential not only to improve the network load balancing, but also in reducing traffic over the longer links.

V. CURRENT RESULTS AND FUTURE DIRECTIONS

Based on the feedback we received from both the simulations and the PlanetLab testings, we are currently designing an improved version of the Julia algorithm. The crux of the improvement is as follows: In the basic Julia algorithm, neighbors are exchanged after the download of each piece. This might create a situation where a high bandwidth node nearby is exchanged for a slower node. We try to prevent this situation using a poker game strategy. The neighbors in our active download set are modeled as a hand of poker: we evaluate the upload performance of the neighbors, as we would evaluate our poker hand. Then, we allow the replacement of any neighbor with a performance below a certain threshold, similar to replacing any subset of poker cards out of our initial hand. We call our modified algorithm the Julia Poker variant.

This strategy is somewhat similar to the BitTorrent probing. In BitTorrent, each node probes for the bandwidth of one neighbor at a time, from among the fixed set of neighbors. If a probed node has a higher upload bandwidth, it is inserted into the active node set, and the lowest performing node is taken out of the active set. However, there are two major differences between the algorithms. In Julia, we allow the replacement of several nodes out of the active set and not just one at

a time. Furthermore, the set of neighbors is not fixed. Nodes are selected from the complete network.

We believe our improved algorithm might work better in practice, since it is more flexible than the BitTorrent selection of nodes, while at the same time preserving the Julia algorithm properties of load balancing in the network. Preliminary simulation results confirming these predictions are shown in figure 4.

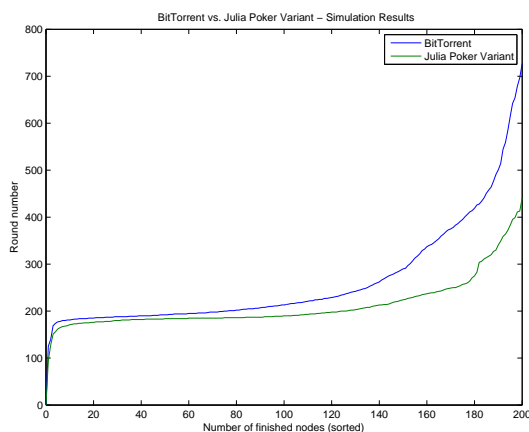


Fig. 4. Finishing download times of BitTorrent vs. Julia Poker Variant using simulation.

Acknowledgements We would like to thank Igal Ermanok for implementing the simulation.

REFERENCES

- [1] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron. Practical locality-awareness for large scale information sharing. 2005.
- [2] 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.
- [3] D. Bickson, D. Malkhi, and D. Rabinowitz. Locality aware content distribution. Technical Report TR-2004-52, 2004.
- [4] 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.
- [5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3-12, 2003.
- [6] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of P2P Economics Workshop*, 2003.
- [7] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturighis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC, 1987*.
- [8] K. C. Ellen W. Zegura and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM 1996, San Francisco, CA*.
- [9] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *In proc. of INFOCOM 2005*, 2005.
- [10] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. "file-sharing in the internet: A characterization of p2p traffic in the backbone. In *Technical Report, University of California, Nov. 2003*.