Optimizing the Reliable Distribution of Large Files within CDNs

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94303
lucy_cherkasova@hp.com

Abstract *Content Delivery Networks (CDNs) provide an efficient support for serving http and streaming media content while minimizing the network impact of content delivery as well as overcoming the server overload problem. For serving the large documents and media files, there is an additional problem of the original content distribution across the CDN edge servers. We propose an algorithm, called ALM-FastReplica, for optimizing replication of large files across the edge servers in CDNs. The original file is partitioned into k subfiles, and each subfile is replicated via a correspondingly constructed multicast tree. Nodes from the different multicast trees use additional cross-nodes connections to exchange their corresponding subfiles such that each node eventually receives an entire file. This new replication method significantly reduces file replication time, up to 5-15 times compared to the traditional unicast (or point-to-point) schema. Since a single node failure in the multicast tree during the file distribution may impact the file delivery to a significant number of nodes, it is important to design an algorithm which is able to deal with node failures. We augment ALM-FastReplica with an efficient reliability mechanism, that can deal with node failures by making local repair decisions within a particular replication group of nodes. Under the proposed algorithm, the load of the failed node is shared among the nodes of the corresponding replication group, making the performance degradation gradual.*

1 Introduction

The amount of streaming media content online increase everyday. Major news companies are among the largest producers of streaming media content. Sports and music video sites add their lion share to popular media content. Video clips of important events (SuperBowl, Olympics, Election'2000, Election'2004) drew unprecedented amounts of traffic. Streaming media can easily become a bandwidth "hog" that overwhelms networks and impacts mission-critical applications unless the special content distribution techniques are used.

Fast and reliable distribution of data from a single source to a large number of receivers located across the Internet has got a lot of attention during last decade. CDNs, e.g. Akamai, employ a dedicated set of machines (edge servers) to distribute content to clients on behalf of the origin server. In this solution, content is replicated at the edge servers (typically via caching) closer to the end user, thereby improving content delivery latency while reducing WAN traffic.

For media files, it is desirable to replicate these files at edge servers in advance, using the so-called *push model*. For large media files it is a challenging, resource-intensive problem. While transferring a large file with individual point-to-point connections from an original server can be a viable solution in the case of limited number of mirror servers (tenths of servers), this method does not scale when the content needs to be replicated across a CDN with thousands of geographically distributed machines.

In our recent work [5], we proposed a new algorithm *FastReplica* for replicating large files in the Internet environment. There are a few basic ideas exploited in *FastReplica*. In order to replicate a large file among k nodes (k is in the range of 10-30 nodes), the original file is partitioned into k subfiles of equal size and each subfile is transferred to a different node in the group. After that, each node sends its subfile to the remaining nodes in the group, and the replication process is iteratively repeated by taking the nodes with already replicated file as the origin nodes. A file transfer in *FastReplica* follows a *store-and-forward* mode, i.e. only after a subfile is received by a node, this node starts sending the corresponding subfile to the subsequent nodes.

A growing number of researchers [3, 4, 7, 9, 12, 8] have advocated the *application level multicast (ALM)* approach, where nodes across the Internet act as intermediate routers and where all multicast related functionality, including group management and packet replication, is implemented at these end systems.

In this work, we propose to combine *FastReplica* ideas with ALM approach for optimizing the replication of the large files within CDN. We call this method *ALM-FastReplica*. The original file is partitioned into k subfiles of equal size, and each subfile is replicated via a correspondingly constructed multicast tree using a *fast-forward* mode, i.e. each node starts a file transfer to the next nodes immediately after receiving the first packet of the corresponding subfile. Nodes from the different multicast trees use additional cross-nodes connections to exchange their corresponding subfiles such that each node eventually receives an entire file. This new replication method significantly reduces file replication time, up to 5-15 times compared to the traditional multiple unicast method. In multiple unicast method, the whole file is transferred in a point-to-point fashion to the first group of nodes, and then each node from the first group initiates file replication to the next group, etc.

Under the "store-and-forward" model, the node failure may only impact the content delivery in the local replication group. However, under the *fast-forward* model, a single node failure in a multicast tree during the file distribution may impact the file delivery to a significant number of nodes. We augment *ALM-FastReplica* with an efficient

reliability mechanism, that can deal with node failures by making local repair decisions within a particular replication group of nodes. Under the proposed algorithm, the load of the failed node is shared among the nodes of the corresponding replication group, making the performance degradation practically unnoticeable.

The rest of the paper explains these ideas in more detail.

2 Related Work

During last years, fast and reliable data distribution from a single source to a large number of receivers located across the Internet has been a topic of many research studies.

One method consists of accessing multiple servers in parallel to reduce downloading time or to achieve fault tolerance. The authors in [14], demonstrate improved response time observed by the client for a large file download through the dynamic parallel access schema to replicated content at mirror servers.

Digital Fountain [1] applies Tornado codes to achieve a reliable data download. In their subsequent work [2], the download times are reduced by having client receive a Tornado encoded file from multiple mirror servers.

In recent years, overlay networks have become an effective alternative to IP multicast for efficient point to multipoint communication across the Internet. An interesting extension for the end-system multicast is introduced in [3], where authors, instead of using the end systems as routers forwarding the packets, propose that the end-systems do actively collaborate in informed manner to improve the performance of large file distribution. The main idea is to overcome the limitation of the traditional service models based on tree topologies where the transfer rate to the client is defined by the bandwidth of the bottleneck link of the path from the server. The authors propose to use additional cross-connections between the end-systems to exchange the complementary content these nodes have already received. However, the authors do not explicitly partition the original document into a set of complementary subfiles for speeding up the file delivery. Their method relies on the assumption that any given pair of end-systems has not received exactly the same content, and these cross-connections between the end-systems can be used to "reconcile" the differences in received content in order to reduce the total transfer time.

BitTorrent [6] represents another approach: it implements peer-to-peer network with centralized tracking of available clients. However, this system has a low resilience to node failures, and could be significantly improved in this direction. Among academic initiatives are Bullet [10] which uses a self-organizing overlay mesh, and SplitStream [4] which uses a collection of superimposed multicast trees resembling *ALM-FastReplica* construction. SplitStream supports the delivery of live media, and tries to optimize the streaming quality of delivered content. It does not attempt to minimize the "download" time.

While CDNs were originally intended for static web content, they have been applied for delivery of streaming media as well. Most of the current work in this direction concentrates on how to improve the media delivery from the edge servers to the end clients. The goal of our paper is to address the content distribution within this infrastructure (and not to the clients of this infrastructure).

3 ALM-FastReplica

Let N_0 be a node which has an original file F. and let$Size(F)$ denote the size of file F in bytes. The problem consists in replicating file F across nodes $N_1, ..., N_n$ while minimizing the overall (both average and maximum) replication time.

Original *FastReplica* algorithm [5] works as follows. Let k be a number of network connections chosen for concurrent transfers between a single node and multiple receiving nodes. The original set of nodes is partitioned into *replication groups*, each consisting of k nodes. File F is divided in k equal subsequent subfiles: $F_1, ..., F_k$. The replication process from N_0 to a first replication group is called *FastReplica in the Small*, and it proceeds as follows.

- *Step 1: Distribution Step*. The originator node N_0 opens k concurrent network connections to nodes $N_1, ..., N_k$ of the first replication group, and sends to each recipient node N_i subfile F_i ($1 \leq i \leq k$).
- *Step 2: Collection Step*. After receiving file F_i, node N_i opens $k-1$ concurrent network connections to remaining nodes in the group and sends subfile F_i to them (i.e. a file transfer is done via a *store-and-forward* mode). At this step, each node N_i has the following set of connections:
  - there are $k-1$ outgoing connections from node N_i: one connection to each node N_j ($j \neq i$) for sending the corresponding subfile F_i to node N_j.
  - there are $k-1$ incoming connections to node N_i: one connection from each node N_j ($j \neq i$) for sending the corresponding subfile F_j to node N_i.

At the end of the *collection* step, each node receives all subfiles $F_1, ..., F_k$ comprising the entire original file F.

After that *FastReplica in the small* is applied iteratively, using nodes $N_1, ..., N_k$ as the origin nodes to replicate file F further to new groups of nodes.

Example. Let $k = 10$. In *three algorithm iterations* ($10 \times 10 \times 10$), the original file can be replicated among 1000 nodes. At each iteration, the replication process follows *FastReplica in the small*, i.e. *the iteration consists of 2 steps*, each used for transferring the $\frac{1}{k}$-th portion of the original file F.

In this work, we propose to combine *FastReplica* schema with *application level multicast* in the following way.

Let k be a number of network connections chosen for concurrent transfers between a single node and multiple receiving nodes. The original set of nodes is partitioned into replication groups, each consisting of k nodes. Let $G_1, ..., G_{k_1}$ be the corresponding replication groups.

Let m be a targeted number of groups comprising a multicast tree (accordingly to previous studies [8], a reasonable value of m may vary in a range of several 10s).

Then replication groups $G_1, ..., G_{k_1}$ are arranged in the special multicast trees $\hat{M}, M^1, ..., M^{m_1}$, each consisting of

m (or less) groups, where $\hat{M}$ is called a *primary* multicast tree, and $M^1, ..., M^{m_1}$ are called the *secondary* multicast trees. To achieve the best performance results, the values m and m_1 should be similar: this will lead to well-balanced multicast trees [1].

File F is divided in k equal subsequent subfiles:

$$F_1, ..., F_k$$

where $Size(F_i) = \frac{Size(F)}{k}$ bytes for each $1 \leq i \leq k$.

A high-level schema of *ALM-FastReplica* is shown in Figure 1. First, *ALM-FastReplica* algorithm replicates file F via the *primary* multicast tree $\hat{M}$. Once groups $\hat{G}_1, ..., \hat{G}_{m_1}$, comprising the *primary* multicast tree $\hat{M}$, **receive subfiles** $F_1, ..., F_k$ **completely** they initiate (independently from each other) transfers of subfiles $F_1, ..., F_k$ to the *secondary* multicast trees $M^1, ..., M^{m_1}$.

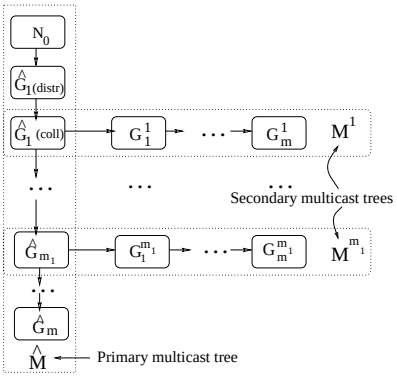

Figure 1: *ALM-FastReplica*: a high-level, overall schema.

Now, we describe *ALM-FastReplica* in more detail.

1). Starting the *primary* multicast tree $\hat{M}$.

Let groups $\hat{G}_1, ..., \hat{G}_m$ comprise the *primary* multicast tree $\hat{M}$. Let $\hat{G}_i = \{N_1^i, ..., N_k^i\},\ 1 \leq i \leq m$.

- *Distribution step*. Originator node N_0 opens k concurrent network connections to nodes $N_1^1, ..., N_k^1$ of group $\hat{G}_1$, and starts sending subfile F_i to the corresponding recipient node N_i^1 $(1 \leq i \leq k)$. This step is represented by box $\hat{G}_1(distr)$ in Figure 1.
- *Collection step*. In group $\hat{G}_1$, each node N_i^1 after receiving first packet of file F_i, immediately starts sending the file F_i to the rest of the nodes in group $\hat{G}_1$ (i.e. a file transfer is done via a *fast-forward* mode). In this way, each node in group $\hat{G}_1$ will be receiving all subfiles $F_1, ..., F_k$ of original file F. This step is represented by box $\hat{G}_1(coll)$ in Figure 1.
- *Group communication step*. Communication between groups $\hat{G}_1$ and $\hat{G}_2$ follows a different file exchange protocol defining another typical communication pattern actively used in the *ALM-FastReplica* algorithm. We denote this step as a *group communication step*. Each node N_i^1 of group $\hat{G}_1$ after receiving first packet of file F_i, immediately starts sending the file F_i to node N_i^2 of group $\hat{G}_2$. After that nodes of group $\hat{G}_2$ perform the *collection step* described above, i.e. each node N_i^2 opens $k-1$ concurrent network connections to the rest of the nodes of group $\hat{G}_2$ for transferring its subfile F_i. In this way, each node of group $\hat{G}_2$ will be receiving all subfiles $F_1, ..., F_k$ of the original file F. The communications between the nodes in groups $\hat{G}_1$ and $\hat{G}_2$ are shown in more detail in Figure 2.

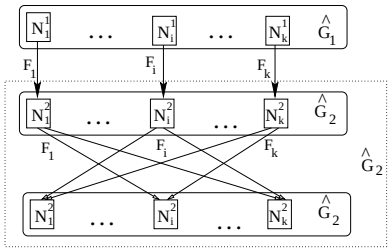

Figure 2: Group communication step.

Similarly, group $\hat{G}_2$ starts communications with group $\hat{G}_3$ using *group communication step* immediately after any node N_i^2 receives the first packet of the corresponding file F_i. This replication procedure continues unrolling through the set of corresponding groups in multicast tree $\hat{M}$ shown in Figure 3.

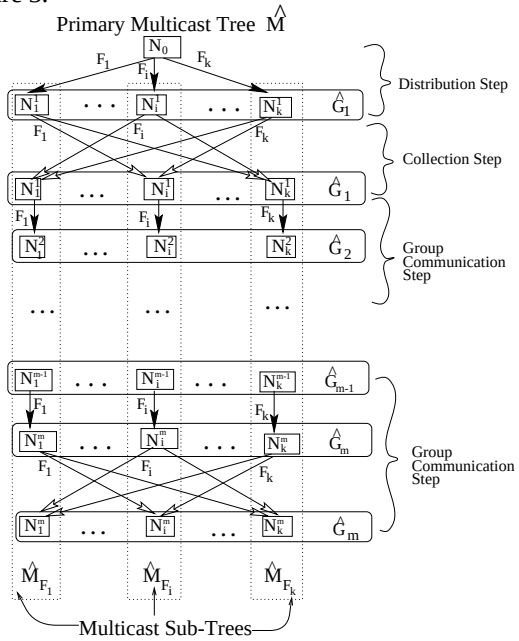

Figure 3: Communications among groups of *primary* multicast tree $\hat{M}$.

In fact, as it is shown in Figure 3, the multicast tree $\hat{M}$ is a collection of k multicast sub-trees $\hat{M}_{F_1}, \hat{M}_{F_2}, ..., \hat{M}_{F_k}$, where each such sub-tree $\hat{M}_{F_i}$ is replicating the corresponding subfile F_i. At the same time, nodes from these different multicast sub-trees use additional cross-connections between their nodes (as shown in Figure 3) to exchange their complementary subfiles.

2). Starting the *secondary* multicast trees $M^1, ..., M^{m_1}$.

- When node N_j^i of $\hat{G}_i$ receives the *entire* subfile F_j in the *primary* multicast tree $\hat{M}$, then it starts transfer-

[1] Depending on the number of nodes in the original replication set, *ALM-FastReplica* may operate by using only a *primary* multicast tree, or may require to employ two-level schema as described in the paper.

ring file F_j to group G_j^i of the *secondary* tree M^i using the *group communication step*. Thus, each group $\hat{G}_i (1 \leq i \leq m_1)$ of the *primary* multicast tree $\hat{M}$ initiates the replication process of subfiles $F_1, ..., F_k$ to the next, *secondary* multicast tree $M_i = \{G_1^i, ..., G_m^i\}$ (see Figure 1). These transfers are asynchronous within the group $\hat{G}_i = \{N_1^i, ..., N_k^i\}$.

This completes the description of *ALM-FastReplica*.

4 Reliable ALM-FastReplica

In this Section, we extend the *ALM-FastReplica* algorithm to be able to deal with node failures. Basic algorithm presented in Section 3 is sensitive to node failures. Under the *fast-forward* model, a single node failure in a multicast tree during the file distribution may impact the file delivery to a significant number of nodes. For example, if node N_1^1 (from group $\hat{G}_1$ in $\hat{M}$) fails during the either *distribution* or *collection* steps described in Section 3 then this event may impact all nodes $N_2^1, ..., N_k^1$ in the group $\hat{G}_1$ because each node depends on node N_1^1 to replicate subfile F_1. A similar situation occurs if a failure of node N_i^1 happens during the group communication step between groups $\hat{G}_1$ and $\hat{G}_2$: this failure may impact all the nodes in the dependent subtree because the nodes in this subtree should receive subfile F_i from node N_i^1.

Thus, it is important to design an algorithm which is able to deal with node failures. It is a non-trivial task because each node in the multicast tree has a wide fan-out (10 or more connections out). Using a naive approach where the tree below the failed node N_{failed} is reconnected to its predecessor N_{prev} will not be a well performing solution. Under such a solution, N_{prev} becomes a severe bottleneck in the whole system since it needs to handle the doubled number of connections.

Reliable *ALM-FastReplica* proposed below efficiently deals with node failures by making the local repair decision within the particular group of nodes. It keeps the main structure of the *ALM-FastReplica* algorithm practically unchanged while adding the desired property of resilience to node failures.

Let us call group $\hat{G}_1$ an *initial group*. We will pay a special attention to group $\hat{G}_1$ and node failures in it, because it is the most vulnerable and crucial group of nodes. Nodes of $\hat{G}_1$ are called *initial nodes*. We will call the rest of the groups *subsequent groups*, and their nodes - *subsequent nodes*.

There are several communication patterns in which node N_i^j might be involved at moment of its failure:

1). If node N_i^j is an *initial* node, i.e. $N_i^j = N_i^1$, it may fail:

- during the *distribution step*, when node N_0 is transferring file F_i to N_i^1. Only node N_0 has subfile file F_i at this point. Since node N_i^1 is failed during the file transfer from N_0 to N_i^1 , node N_0 is aware of node N_i^1 failure.
- during the *collection step*, when node N_i^1 has received first packet of file F_i and started to transfer it to the remaining nodes in group $\hat{G}_1$.
- during the *group communication step* in the first multicast tree $\hat{M}$, i.e. when node N_i^1 is transferring file F_i to the nodes of group $\hat{G}_2$.
- during the *group communication step* to the next multicast tree M^1, i.e. when node N_i^1 is transferring file F_i to the nodes of group G_1^1 as shown in Figure 1. The crucial difference of the node failure at this step is that any node in group $\hat{G}_1$ already has received subfile F_i.

2). If node N_i^j is a *subsequent* node, it may fail during the *group communication step* when node N_i^j of group G_j is transferring file F_i to the nodes of the next group G_{j+1}.

In the reliable *ALM-FastReplica* algorithm, node N_i^j of group G_j sends *heart-beat messages* to node N_i^{j-1} of group G_{j-1}. These heart-beat messages are augmented with additional information on the corresponding algorithm step and the current replication list of nodes corresponding to this step. This information is necessary because of the asynchronous nature of the *ALM-FastReplica* algorithm. For example, while some of the nodes of group $\hat{G}_1$ can perform a file transfer in the *primary* multicast tree $\hat{M}$ (i.e. they are still replicating their subfiles to the corresponding nodes of group $\hat{G}_1$), some "faster" nodes of the same group $\hat{G}_1$ might already have started a file transfer to the next multicast tree M^1 (i.e. they are replicating their subfiles to the corresponding nodes of group G_1^1 as shown in Figure 1).

Thus in case of node failure, it is important to know:

- which particular node in the group has failed;
- whether the node is *initial* or not;
- whether the current step of the algorithm is the *distribution*, *collection*, or *group communication* step;
- which multicast tree, group and set of receiving nodes are impacted as a result of this failure.

There are different repair procedures depending on the circumstances under which the node failure occurred:

- The *initial* node N_i^1 of group $\hat{G}_1$ fails during *distribution*, *collection*, or *group communication* step in the *primary* multicast tree $\hat{M}$.

  In this case, node N_0 is either aware of the node N_i^1 failure or receives a message about this from the heartbeat group $\hat{G}_1$. Since node N_0 is the root of the overall replication procedure, to avoid a single point of failure it has a *buddy-node* $\hat{N}_0$ with mirrored information and data. Node N_0 sends a message to $\hat{N}_0$ to open $k-1$ network connections to the rest of the nodes in group $\hat{G}_1$ for sending the missing file F_i to each node in the group.

- Node N_i^j fails during the *group communication step*, where node N_i^j is either a *subsequent* node, or an *initial* node performing the *group communication step* to *secondary* multicast tree.

Figure 4 depicts a failure of node N_i^j in group G_j while it was transferring file F_i to the rest of the nodes in group G_j and further to node N_i^{j+1} of group G_{j+1} at the *group communication step* of the algorithm.

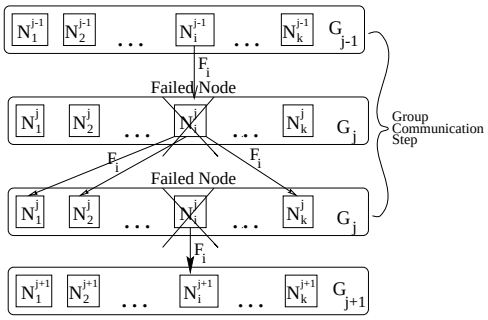

Figure 4: Node N_i^j failure during *group communication step*.

The repair procedure for node N_i^j is shown in Figure 5. Once node N_i^{j-1} realized that node N_i^j has failed, node N_i^{j-1} conveys this information to the rest of the nodes in group G_{j-1}. The nodes in group G_{j-1} share the additional load of transferring file F_i to the nodes of group $G_j \setminus N_i^j$ as shown in Figure 5. Node N_i^{j-1} transfers file F_i to node N_i^{j+1} of group G_{j+1}.

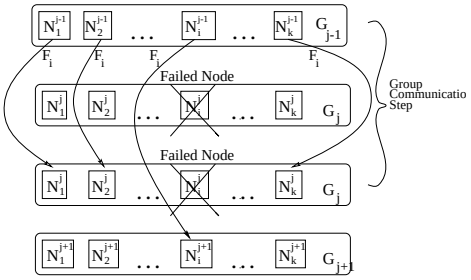

Figure 5: Repair procedure for node N_i^j failed during *group communication step*.

The distribution lists are the same for all the nodes of group G_j, and after the "repair" step, the *ALM-FastReplica* algorithm proceeds in the usual way for the entire subtree originated in group G_j. The nodes in group G_{j-1} continue to share the additional load of transferring file F_i to the next, *secondary* multicast tree originated at a group G^j or until node N_i^j is repaired. Since the load of failed node N_i^j is shared among k nodes of group G^{j-1}, the performance degradation is gradual for the repaired portion of the distribution tree.

5 Performance Evaluation

Let $Time^i(F)$ denote the transfer time of file F from the original node N_0 to node N_i. We use *transfer time* and *replication time* interchangeably in the text. In our study, we consider the following two performance metrics:

- *Average replication time:*
$$Time_{aver} = \frac{1}{n}\sum_{i=1}^{n} Time^i(F)$$

- *Maximum replication time:*
$$Time_{max} = max\{Time^i(F)\}, i \in \{1, \cdots, n\}$$

$Time_{max}$ reflects the time when all the nodes in the replication set receive a copy of the original file, and the primary goal of *ALM-FastReplica* is to minimize the maximum replication time. However, we are also interested in understanding the impact of *ALM-FastReplica* on the average replication time $Time_{aver}$.

To analyze performance benefits of *ALM-FastReplica*, let us consider a case with 1000 replication nodes, where each node can open $k = 10$ concurrent connections, i.e. there are 100 groups, each consisting of $k = 10$ nodes.

Let *Multiple Unicast* denote a schema that works iteratively in the following way. First, it transfers the *entire* file F from the original node to a first group of 10 nodes by simultaneously opening $k = 10$ point-to-point concurrent network connections. Then, it uses these 10 nodes (when they receive the entire file F) as the origin nodes and repeats the same procedure to the new groups of nodes, etc.

The original *FastReplica* algorithm works iteratively too. File F is divided in $k = 10$ equal subsequent subfiles and is replicated to a first group of 10 nodes. Once they receive all the subfiles, comprising file F, *FastReplica* uses these 10 nodes as the origin nodes with file F and repeats the same procedure again, etc. Thus in *three iterations*, file F can be replicated among 1000 nodes. Each iteration consists of 2 steps (*distribution* and *collection*), each used for transferring the $\frac{1}{10}$-th portion of the original file F.

The *ALM-FastReplica* algorithm operates as described in Section 3 using a *primary* multicast tree consisting of 10 replication groups, and 9 *secondary* multicast trees, each also consisting of 10 replication groups.

Let BW denote a *bandwidth matrix*, where $BW[i][j]$ reflects the available bandwidth of the path from node N_i to node N_j (as measured at some time T), and let *Var* be the ratio of maximum to minimum available bandwidth along the paths participating in the file transfers. We call *Var* a *bandwidth variation*.

In our analysis, we consider the bandwidth matrix BW to be populated in the following way:

$$BW[i][j] = B \times random(1, Var),$$

where function $random(1, Var)$ returns a random integer var: $1 \leq var \leq Var$.

While it is a simplistic model, it helps to reflect a realistic situation, where the available bandwidth of different links can be significantly different. We will call this model a *uniform-random model*.

To perform a sensitivity analysis of how *ALM-FastReplica* performance depends on a bandwidth variation of participating paths, we experimented with a range of different values for *Var* between 1 and 10. When $Var = 1$, it is an *idealistic setting*, where all of the paths are homogeneous and have the same bandwidth B (i.e. no variation in bandwidth). When $Var = 10$, the network paths between the nodes have highly variable available bandwidth with a possible difference of up to 10 times.

Using the *uniform-random* model and its bandwidth matrix BW, we computed the average and maximum file replication times under *Multiple Unicast*, *FastReplica*, *ALM-FastReplica* for 1000 nodes in the replication set. After that, we derived the relative speedup of the file replication time under *ALM-FastReplica* compared to the replication time under the *Multiple Unicast* and *FastReplica* strategies. Relative speedup of *average* file replication time is shown in Figure 6.

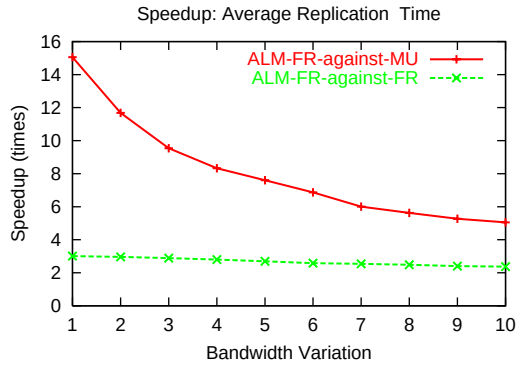

Figure 6: *Uniform-random model*: speedup in average file replication time under *ALM-FastReplica*.

For *Var=1*, there is no variation in bandwidth, and all the paths have the same bandwidth B. Under this *idealistic setting*, *ALM-FastReplica* outperforms *FastReplica* – three times, and *Multiple Unicast* – fifteen times as shown in Figure 6. It can be explained in the following way.

The *ALM-FastReplica* replication process consists of two steps: 1) *primary* multicast tree and 2) *secondary* multicast trees, each used for transferring the $\frac{1}{10}$-th portion of the original file F.

The *FastReplica* replication process consists of three iterations. At each iteration, the replication process follows *FastReplica in the small*, i.e. the iteration consists of 2 steps, each used for transferring the $\frac{1}{10}$-th portion of the original file F. Thus overall *FastReplica* performs 6 steps, each used for transferring the $\frac{1}{10}$-th portion of the original file F. Thus, the replication time under *ALM-FastReplica* is 3 times better than under *FastReplica*: $\frac{6}{10} \ / \ \frac{2}{10} = 3$.

Finally, the *Multiple Unicast* replication process consists of three iterations, each used for transferring the *entire* file F. Thus, the replication time under *ALM-FastReplica* is 15 times better than under *Multiple Unicast*: $3 \ / \ \frac{2}{10} = 15$.

While the performance benefits of *ALM-FastReplica* are decreasing for higher variation of bandwidth of participating paths, it still remains very efficient, outperforming *FastReplica* – more than two times, and *Multiple Unicast* – more than five times in average replication time under bandwidth variation of 10.

Maximum replication time under *ALM-FastReplica* is three times better than under *FastReplica*, and fifteen times better than under *Multiple Unicast* independent of the values of bandwidth variation (intuitively, these are improvements of the worst path in *uniform-random* model, and the comparison of replication time for a worst case scenario under different strategies is similar to *idealistic setting* case).

6 Conclusion

In recent years, the Internet services has moved from an architecture where data objects are located at a single origin server to the architecture where objects are replicated across multiple, geographically distributed servers. Client requests for content are redirected to a best-suited replica rather than the origin server. While CDNs were originally intended for static web content, they have been applied for delivery of streaming media as well. However, for large media files, the replication process across this distributed network of servers is a challenging, resource-intensive problem on its own.

In this work, we propose a new algorithm, called *ALM-FastReplica*, for an efficient and reliable replication of media files in the Internet environment. This algorithm combines the original *FastReplica* ideas with ALM approach for significantly reducing the replication time of the large files within CDN. We augment *ALM-FastReplica* with an efficient reliability mechanism, that can deal with node failures by making local repair decisions within a particular replication group of nodes. Proposed reliability algorithm is constructed in such a way, that in the repaired portion of the distribution tree, the performance degradation is gradual, since the load of the failed node is shared among the nodes of the "preceding" replication group.

References

[1] J. Byers, M .Luby, M. Mitzenmacher, A. Rege. A Digital Fountain approach to reliable distribution of bulk data. *Proc. of ACM SIGCOMM*, 1998.

[2] J. Byers, M. Luby, M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using Tornado codes to speedup downloads. *Proc. of IEEE Infocom*, 1999.

[3] J. Byers, J. Considine, M. Mitzenmacher, S. Rost. Informed content delivery across adaptive overlay networks. *Proc. of ACM SIGCOMM*, 2002.

[4] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. *Proc. of ACM SOSP*, 2003.

[5] L. Cherkasova, J. Lee. FastReplica: Efficient Large File Distribution within Content Delivery Networks. *Proc. of the 4th USENIX Symposium on Internet Technologies*, March, 2002.

[6] B. Cohen. BitTorrent, 2003. *http://bitconjurer/BitTorrent.org*

[7] Y. Chu, S. Rao, H. Zhang. A case for end system multicast. *Proc. of ACM SIGMETRICS*, June, 2000.

[8] Y. Chu, S. Rao, S. Seshan, H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. *Proc. of ACM SIGCOMM*, 2001.

[9] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, J. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. *Proc. of the 4th Symp. on Operating System Design and Implementation (OSDI)*, 2000.

[10] D. Kostic, A. Rodriguez, J. Albrecht, A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. *Proc. of the 19th ACM SOSP'2003*.

[11] T. Nguyen and A. Zakhor. Distributed Video Streaming over the Internet. *Proc. of SPIE Conference on Multimedia Computing and Networking*, San Jose, CA, 2002.

[12] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: an application level multicast infrastructure. *Proc. of USITS'2001*.

[13] Justin Ridge, Fred W. Ware and Jerry D. Gibson. Multiple Descriptions, Error Concealment, and Refined Descriptions for Image Coding. Proc. Second Annual UCSD Conference on Wireless Communications, 1999.

[14] P. Rodriguez, A. Kirpal, and E. W. Biersack. Parallel-access for mirror sites in the Internet. *Proc. of IEEE Infocom*, 2000.