Optimizing Data Management in Grid Environments

Antonis Zissimos, Katerina Doka, Antony Chazapis, Dimitrios Tsoumakos, and Nectarios Koziris

National Technical University of Athens
School of Electrical and Computer Engineering
Computing Systems Laboratory
{azisi,katerina,chazapis,dtsouma,nkoziris}@cslab.ece.ntua.gr

Abstract. Grids currently serve as platforms for numerous scientific as well as business applications that generate and access vast amounts of data. In this paper, we address the need for efficient, scalable and robust data management in Grid environments. We propose a fully decentralized and adaptive mechanism comprising of two components: A Distributed Replica Location Service (*DRLS*) and a data transfer mechanism called *GridTorrent*. They both adopt Peer-to-Peer techniques in order to overcome performance bottlenecks and single points of failure. On one hand, DRLS ensures resilience by relying on a Byzantine-tolerant protocol and is able to handle massive concurrent requests even during node churn. On the other hand, GridTorrent allows for maximum bandwidth utilization through collaborative sharing among the various data providers and consumers. The proposed integrated architecture is completely backwards-compatible with already deployed Grids. To demonstrate these points, experiments have been conducted in LAN as well as WAN environments under various workloads. The evaluation shows that our scheme vastly outperforms the conventional mechanisms in both efficiency (up to 10 times faster) and robustness in case of failures and flash crowd instances.

1 Introduction

One of the most critical components in Grid systems is the data management layer. Grid computing has attracted several data-intensive applications in the scientific field, such as bioinformatics, physics or astronomy. To a great extent, these applications rely on analysis of data produced by geographically disperse scientific devices such as sensors or satellites etc. For example, the Large Hadron Collider (LHC) project at CERN [1] is expected to generate tens of terabytes of raw data per day that have to be transferred to academic institutions around the world, in seek of the Higgs boson. Apart from that, business applications manipulating vast amounts of data have lately started to invade Grid environments. *Gredia* [2] is an EU-funded project which proposes a Grid infrastructure for sharing of rich multimedia content. To motivate this approach, let us consider

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 497–512, 2009.

the following scenario: News agencies have created a joint data repository in the Grid, where journalists, photographers, editors, etc can store, search and download various news content. Assume that just minutes after a breaking news-flash (e.g., the riots in Athens), a journalist on scene captures a video of the protests and uploads it on the Grid. Hundreds of journalists and editors around the world need to be able to quickly locate and efficiently download the video in order to include it in their news reports. Thus, it is imperative that, apart from optimized data transfer, such a system should be able to cope with high request rates – to the point of a flash crowd.

Faced with the problem of managing extremely large scale datasets, the Grid community has proposed the Data Grid architecture [13], defining a set of basic services. The most fundamental of them are the Data Transfer service, responsible for moving files among grid nodes (e.g., *GridFTP* [7]), the Replica Location service (*RLS*), which keeps track of the physical locations of files and the Optimization service, which selects the best data source for each transfer in terms of completion time and manages the dynamic replica creation/deletion according to file usage statistics.

However, all of the aforementioned services heavily rely on centralized mechanisms, which constitute performance bottlenecks and single points of failure: The so far centralized RLS can neither scale to large numbers of concurrent requests nor keep pace with frequent updates performed in highly dynamic environments. GridFTP fails to make optimal use of all bandwidth resources in cases where the same data must be transferred to multiple sites and does not automatically maximize bandwidth utilization. Even when using multiple parallel TCP channels, a manual configuration is required. Most importantly, GridFTP servers face the danger of collapsing under heavy workload conditions, making critical data unavailable.

In this paper, we introduce a novel data management architecture which integrates the location service with data transfer under a fully distributed and adaptive philosophy. Our scheme comprises of two parts that cooperate to efficiently handle multiple concurrent requests and data transfer: The *Distributed Replica Location Service (DRLS)* that handles the locating of files and *GridTorrent* that manages the file transfer and related optimizations. This is pictorially shown in Figure 1.

DRLS utilizes a set of nodes that, organized in a DHT, equally share the replica location information. The unique characteristic of the DRLS is that, besides the decentralization and scalability that it offers, it fully supports updates on the multiple sites of a file that exist in the system. Since in many dynamic applications data locations change rapidly with time, our Byzantine-tolerant protocol guarantees consistency and efficiently handles updates on the various data locations stored, unlike conventional DHT implementations. GridTorrent is a protocol that, inspired by BitTorrent, focuses on real-time optimization of data transfers on the Grid, fully supporting the induced security mechanisms. Based on collaborative sharing, GridTorrent allows for low latency and maximum bandwidth utilization, even under extreme load and flash crowd conditions. It allows

transfers from multiple sites to multiple clients and maximizes performance by piece exchange among the participants. A very important characteristic of the proposed architecture is that it is designed to interface and exploit well-defined and deployed Data Grid components and protocols, thus being completely backwards compatible and readily deployable. This work includes an experimental section that includes a real implementation of the system and results over both LAN and WAN environments with highly dynamic and adverse workloads.

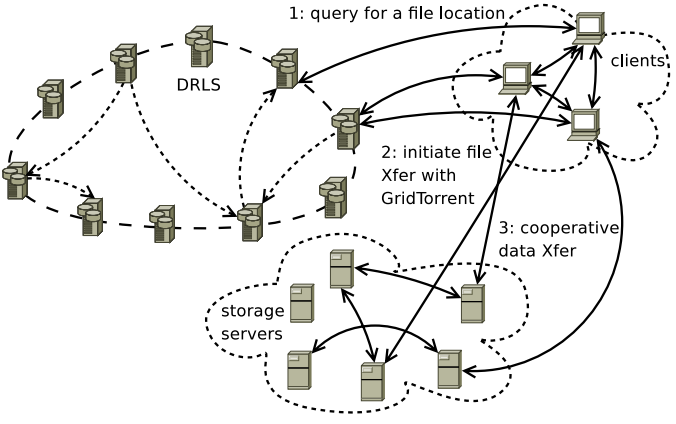

Fig. 1. Pictorial description of the proposed architecture and component interaction. Although DRLS nodes and Storage Servers appear to be a separate physical entity, it is possible to coexist in order to exploit all the available resources.

2 Current Status

In this section we overview the related work in the area of data management. We first go through existing practices for the Replica Location and Data Transfer services. Next, we present a brief description of the BitTorrent protocol, which is the basis of the proposed GridTorrent mechanism and we finally mention other relevant data transfer mechanisms.

2.1 Locating Files

Centralized Catalog and Giggle. In Grid environments, it is common to maintain local copies of remote files, called *replicas* [23] to guarantee availability and reduce latencies. To work with a file, a Grid application first asks the RLS to locate corresponding *replicas* of the requested item. This translates to a query towards the *Replica Catalogue*, which contains mappings between *Logical File Names* (LFNs) and *Physical File Names* (PFNs). If a local replica already exists the application can directly use it, otherwise it must be transferred to the local node. This initial architecture posed limitations to the scalability and resilience of the system. Efforts on distributing the catalog resulted in the most widespread

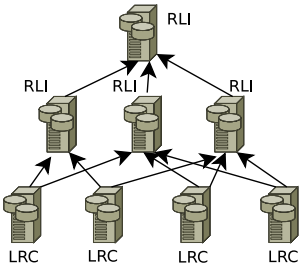

Fig. 2. Replica Location Service deployment scenario with Giggle

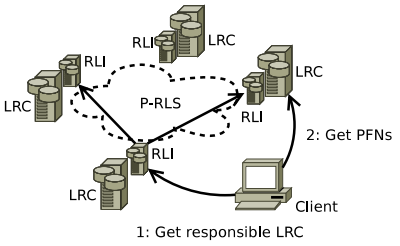

Fig. 3. Replica Location Service deployment scenario with P-RLS

solution currently deployed on the Grid, the Giggle Framework [14]. To achieve distribution, Giggle proposes a two-tier architecture, comprising of the *Local Replica Catalogs* (*LRC*s), which map LFNs to PFNs across a site and the *Replica Location Indices* (*RLI*s), which map LFNs to LRCs (Figure 2).

Distributed Replica Location Service (DRLS). Still, the centralized nature of the catalogs remains the bottleneck of the system, when the number of performed searches increases. Furthermore, the updates in the LRCs induce a complex and bandwidth-consuming communication scheme between LRCs and RLIs. To this end, in [12] we proposed a RLS based on a Distributed Hash Table (DHT). The underlying DHT is a modified Kademlia peer-to-peer network that enables mutable storage. In this work, we enhance our solution by exploiting *XOROS* [11], a DHT that provides a Byzantine-tolerant protocol for serializable data updates directly at the peer-to-peer level. In this way, we can store static information such as file properties with the traditional distributed hash table put/get mechanism, as well as dynamic information such as the actual LFN to PFN mappings with an update mechanism that ensures consistency.

Related Work. Peer-to-peer overlay networks and corresponding protocols have already been incorporated in other RLS designs. In [10], Min Cai *et al.*, have replaced the global indices of Giggle with a Chord network, producing a variant of Giggle called P-RLS. A Chord topology can tolerate random node joins and leaves, but does not provide data fault-tolerance by default. The authors choose to replicate the distributed RLI index in the *successor set* of each *root node* (the node responsible for storage of a particular mapping), effectively reproducing Kademlia's behavior of replicating data according to the replication parameter κ. In order to update a specific key-value pair, the new value is inserted as usual, by finding the *root node* and replacing the corresponding value stored there and at all nodes in its *successor set*. While there is a great resemblance to this design and the one we propose, there is no support for updating key-value pairs directly in the peer-to-peer protocol layer. It is an open question how the P-RLS design would cope with highly transient nodes. Frequent joins and departures in the Chord layer would require nodes continuously exchanging

key-value pairs in order to keep the network balanced and the replicas of a particular mapping in the correct successors. Our design deals with this problem, as the routing tables inside the nodes are immune to participants that stay in the network for a very short amount of time. Moreover, our protocol additions to support mutable data storage are not dependent on node behavior; the integrity of updated data is established only by relevant data operations. Finally, the P-RLS approach retains the two-tier Giggle architecture, since the actual LFN to PFN mappings are still kept in Local Replica Catalogs imposing a bottleneck for the whole system with no support for load-balancing and failover mechanisms. In another variant of an RLS implementation using a peer-to-peer network [21], all replica location information is organized in an unstructured overlay and all nodes gradually store all mappings in a compressed form. This way each node can locally serve a query without forwarding requests. Nevertheless, the amount of data (compressed or not) that has to be updated throughout the network each time, can grow to such a large extent, that the scalability properties of the peer-to-peer overlay are lost. In contrast to other peer-to-peer RLS designs, we envision a service that does not require the use of specialized servers for locating replicas. According to our design, a lightweight DHT-enabled RLS peer can even run at every node connected to the Grid.

2.2 Transferring Files

The GridFTP Protocol. The established method for data transfer in the Grid is *GridFTP* [7], a protocol defined by the Global Grid Forum and adopted by the majority of the existing middleware. GridFTP extends the standard FTP, including features like the *Grid Security Infrastructure* (GSI) [17] and third-party control and data channels. A more distributed approach of the GridFTP service has lead to the *Globus Stripped GridFTP* protocol [8], included in the current release of the Globus Toolkit 4 [3]. Transfers of data striped or interleaved across multiple servers, partial file transfers and parallel data transfers using multiple TCP streams are some of the newly added features.

The GridTorrent Approach. Yet, the GridFTP protocol is still based on the client-server model, inducing all the undesirable characteristics of centralized techniques, such as server overload, single points of failure and the inability to cope with flash crowds. We argue that large numbers of potential downloaders together with the well-documented increase in the volume of data by orders of magnitude stress the applicability of this approach. We propose a replica-aware algorithm based on the P2P paradigm, through which data movement services can take advantage of multiple replicas to boost aggregate transfer throughput. In our previous work [27] there were made some preliminary steps towards this direction. A first GridTorrent prototype was implemented and one could use the Globus RLS and various GridFTP storage servers to download a file, as well as exploit other simultaneous downloaders, thus making a first step towards cooperation. Nevertheless, a core component of every Grid Service, the Globus Security Infrastructure (GSI) wasn't integrated with our previous prototype.

Furthermore, in torrent-like architectures like GridTorrent there is the inherent problem of not being able to upload a file unless there are downloaders interested in the specified file. To tackle this problem we introduce the GridTorrent's control channel, a separate communication path that can be used to issue commands to remote GridTorrent servers. Thus, in order to upload a file several GridTorrent servers are automatically notified and after the necessary authentication and authorization phases, the file is uploaded to multiple servers simultaneously and more efficiently. There is no need for the user to issue another set of commands for replication, because this is handled by GridTorrent. Finally, in order to scale to larger deployments our prototype is integrated with the aforementioned DRLS. In the present work, we extend GridTorrent and propose a complete architecture which can be directly deployed in a real-life Grid environment and integrate with existing Grid services.

The BitTorrent Protocol. Our work as well as other related work on the area rely on the BitTorrent protocol [15]. BitTorrent is a peer-to-peer protocol that allows clients to download files from multiple sources while uploading them to other users at the same time, rather than obtaining them from a central server. Its goal is to reduce the download time for large, popular files and the load on servers that serve these files. BitTorrent divides every file in *piece* and each piece in *blocks*. Clients find themselves through a centralized service called the *tracker* and can exploit this fragmentation by simultaneously downloading blocks from many sources. Useful file information is stored in a *metainfo* file, identified by the extension `.torrent`. Peers are categorized in *seeds* when they already have the whole file and *leechers* when they are still downloading pieces. The latest version of the BitTorrent client [4] uses a Distributed Hash Table (DHT) for dynamically locating the tracker responsible for each file transaction. Note that, in contrast to the Data Management architecture presented here, BitTorrent does not yet use a DHT for storing and distributing file information and metadata. The corresponding .torrent files still have to be downloaded from a central repository, or manually exchanged between users. The data transfer component of our architecture, GridTorrent, enhances the BitTorrent protocol with new features in order to make it compatible with existing Grid architectures. Moreover, new functionality is added, so as to be able to instruct downloads to remote peers. Finally, the tracker, which constitutes a centralized component of the BitTorrent architecture is replaced by DRLS, eliminating possible performance bottlenecks and single points of failure.

Related Work Using BitTorrent. A related work that is based in torrent-like architecture for data transfers in Grid environments can be found in GridTorrent Framework [18], which cites our previous work and therefore should not be confused our proposed architecture. The authors of GridTorrent Framework focus on a centralized tracker to provide information for the available replicas, but also use the tracker to impose security policies for data access. Their work also extend to the exploitation of parallel TCP streams between two single peers in order to surpass the limitations of the TCP window algorithm and saturate high

bandwidth links. Nevertheless, the Framework's centralized design suffers of all the undesirable characteristics of centralized techniques, while the lack of integration with standardized Grid components remains a substantial disadvantage. A similar work is presented in [25], where the authors compare BitTorrent to FTP for data delivery in Computational Desktop Grids, demonstrating that the former is efficient for large file transfers and scalable when the number of nodes increases. Their work is concentrated in application environments like SETI@Home [16], distributed.net [5] and BOINC [9] where methods like cpu scavenging are used to get temporary resources from Desktop computers. In contrast to GridTorrent, their prototype uses centralized data catalog and repository, fails to communicate with standard Grid components like GridFTP and RLS, lacks the support of Globus Security Infrastructure and doesn't tackle the problem of efficient file upload in multiple repositories.

Other Data Transfer Mechanisms. The efficient movement of distributed volumes of data is a subject of constant research in the area of distributed systems. Various techniques have been proposed, apart from the ones mentioned above, centralized or in the context of the peer-to-peer paradigm. Kangaroo [24] is a data transfer system that aims at better overall performance by making opportunistic use of a chain of servers. The Composite Endpoint Protocol [26] collects high-level transfer data provided by the user and generates a schedule which optimizes the transfer performance by producing a balanced weighting of a directed graph. Nevertheless, the aforementioned models remain centralized. Slurpie [22] follows a similar approach to BitTorrent, as it targets bulk data transfer and makes analogous assumptions. Nonetheless, unlike BitTorrent, it does not encourage cooperation.

3 GridTorrent

GridTorrent, a peer-to-peer data transfer approach for Grid environments, was initially introduced in [27]. Based on BitTorrent, GridTorrent allows clients to download files from multiple sources while uploading them to other users at the same time, rather than obtaining them from a central server. Using BitTorrent terminology, GridTorrent creates a swarm where leechers are users of the Grid downloading data and seeds are storage elements or users sharing their data in the Grid. The cooperative nature of the algorithm ensures maximum bandwidth utilization and its tit-for-tat mechanism provides scalability in heavy load conditions or flash crowd situations. More specifically, GridTorrent exploits existing infrastructure since GridFTP repositories can be used as seeds with other peers downloading from them using the GridFTP partial file transfer capability. The *.torrent* file used in BitTorrent is replaced by the already existing RLS. In order to start a file download only the file's unique identifier (*UID*) is required, which is actually the content's digest. The rest of the information can be extracted from the RLS using this UID. Finally, GridTorrent makes the BitTorrent's tracker service obsolete and integrates its functionality in the RLS. Therefore, all the

peers that participate in a GridTorrent swarm are also registered in the RLS, so that they are able to locate each other. In the following paragraphs we analyze the further enhancements we have developed in GridTorrent.

3.1 Security

In a Grid environment, only authenticated users are considered trustworthy of serving or downloading file fragments. Moreover, encryption is provided for the transfer of sensitive information. In order to guarantee security, our data transfer mechanism implements the Globus Grid Security Infrastructure (GSI). Currently, GridTorrent deploys the standard GSI mechanisms, in terms of authentication, integrity and encryption. A Java TCP socket is created and wrapped, along with the host credentials, as a grid-enabled socket. This is performed when the plain socket passes through the createSocket method of the GssSocketFactory of the globus GSI API. Thus, an appropriate socket is created, with respect to the input parameters that enable encryption, message integrity, peer authentication or none of the above, according to the user's preferences.

3.2 Control Channel

In GridTorrent, peers communicate with each other and exchange information regarding the current file download according to the protocol. A novel feature of GridTorrent, not found in BitTorrent protocol, is the ability of a peer to issue commands to remote peers. We call this feature *control channel*, because it is similar to the GridFTP's control channel. This feature overcomes the BitTorrent disadvantage of not being able to upload data before another peer is interested to download them, which is common practice for a peer-to-peer network, but not applicable to Grid environments. In detail, the GridTorrent control channel supports the following commands:

`Start.` [UID] [RLS] Starts downloading the file with the given UID, getting publishing information from the given RLS.

`Start.` [$filename$] [RLS] Starts sharing the existing file determined by the given local filename. RLS will be used for publishing information regarding the download.

`Stop.` [UID] Stops an active file download. Takes as a parameter the UID of the file to stop downloading.

`Delete.` [$filename$] Deletes a local file.

`List.` Lists all active file downloads of the node.

`Get.` [UID] Gets statistics about an active file download regarding messages exchanged and data transfer throughput. Takes the UID of the file as a parameter.

`Shutdown.` Shuts down the GridTorrent peer.

4 Replica Location Service

The RLS used in GridTorrent stores two types of metadata: static information (file properties) and dynamic information (peers that have the file or part of it).

In our design, we select a set of attributes required to initiate a torrent-like data transfer. Therefore, the file properties stored in the RLS are the following:

Logical filename (LFN): This is the name of the stored file. This name is supplied by the user to identify his file.
File size: The total size of the file in bytes.
File hash type: The type of the hashing used to identify the whole file data. Hashing is enabled in this level to ensure data consistency.
File hash: The actual file data hash. It is also used as a UID for each file.
Piece length: The size of each piece in which the file is segmented. The piece is the smallest fraction of data that is used for validating and publishing purposes. Upon a complete piece download and integrity check, other peers are informed of the acquisition.
Piece hash type: The type of the hashing used to identify each piece of the file. Hashing is enabled in this level to facilitate partial download and resume download operations.
Piece hash: The actual piece data hash. All the hashes of all the pieces are concatenated starting from the first piece.

Besides the file properties, the RLS also stores a list of all the physical locations where the file is actually stored. This is described by a physical filename (PFN). A physical filename has the following form:

```
protocol://fqdn:port/path/to/file
```

where `protocol` is the one that is used for the data transfer. Currently the supported protocols are `gsiftp` (GridFTP) and `gtp` (GridTorrent). The fully qualified domain name `fqdn` is the DNS registered name of the peer and it is followed by the peer's local path and the local filename.

4.1 Distributed RLS

RLS as a core Grid service must use distribution algorithms with unique scalability and fault-tolerance properties–assets already available by peer-to-peer architectures. To this end, in [12] we proposed a Replica Location Service based on a Distributed Hash Table (DHT). The underlying DHT is a modified Kademlia peer-to-peer network that enables mutable storage. We enhance this work by exploiting the XOR Object Store (XOROS) [11], a DHT that provides serializable data updates to the primary replicas of any key in the network. XOROS uses a Kademlia [19] routing scheme, along with a modified protocol for inserting and looking up values, that accounts for dynamic or Byzantine behavior of overlay participants. The put operation allows either an in-place update, or a read-modify-write via a single, unified transaction, that consists of a mutual exclusion mechanism and an accompanying value propagation step. GridTorrent has a modular architecture that enables the use of different types of Replica Location Service per swarm. More specifically, when a user initiates a file transfer, he must also supply the RLS URL, which has the following form:

```
protocol://fqdn:port
```

Table 1. Security overhead in the overall file transfer

configuration	mean time (sec)	overhead
authentication	43,3	0%
authentication + integrity check	44,3	2%
authentication + encryption	55,3	27%

Currently the supported protocols are `rls` (Globus RLS) and `drls` (Distributed RLS based on XOROS), so GridTorrent parses the URL to load the corresponding RLS implementation. One advantage of the above modification is the use of already implemented features to model our solution, preserving the backwards compatibility with the existing Grid Architecture. Therefore, the proposed changes in the current Grid Architecture not only enhance the performance of data transfers, but also seamlessly integrate with the current state-of-the-art in Grid Data Management.

5 Implementation and Experimental Results

Our GridTorrent prototype implementation is entirely written in Java. The GridTorrent client has bindings with Globus Toolkit 4 libraries [3] and exploits the GridFTP client API, the Replica Location Service API and the Grid Security Infrastructure API. These bindings enrich our prototype with the abilities to use existing grid infrastructure, such as data stored in GridFTP servers, metadata stored in Globus RLS and x509 certificates that are already issued to users and services for authentication, authorization, integrity protection and confidentiality. For the experiments we started GridTorrent to a number of physical nodes and issued remote requests through the control channel, to initiate and monitor the overall file transfer.

5.1 GridTorrent Security and Fault-Tolerance Performance

We first test the effect that Grid Security has in the overall data transfer process by monitoring the time needed for the transfer of a 128MB file. We distinguish three different configurations for Globus GSI:

Authentication only: This is a simple configuration where both sides need to present a valid x509 certificate signed from a Certificate Authority that is mutually trusted.

Integrity check: In this configuration besides the mutual authentication, the receiver verifies all messages to prevent man-in-the-middle attacks.

Encryption: This is the most secure configuration, where apart form mutual authentication and integrity check, every message is also encrypted.

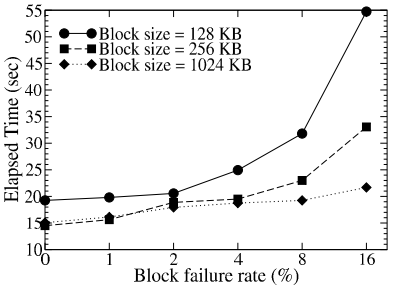

Fig. 4. Average time of completion over various number of failure rates and block sizes

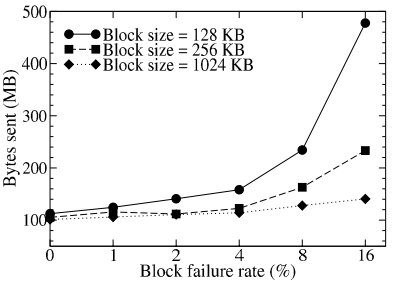

Fig. 5. Average size of uploaded data from leechers only over various number of failure rates and block sizes

The test is executed 100 times between a pair of peers (different each time) inside the same LAN. As shown in Table 1, only the Globus GSI configuration that enables encryption has considerable (about 30%) cost on the file transfer latency. This overhead is natural, because when encryption is enabled every message is duplicated in memory and parsed by a cpu-intensive cryptographic algorithm.

We continue our experiments by testing GridTorrent's tolerance in an error-prone network. For this purpose we use a single server acting as seed for a file of 128MB and 16 clients that simultaneously download the file. After extensive testing we have tuned GridTorrent to use a piece size of 1024KB. In GridTorrent, just like BitTorrent, hashes are kept in a per piece basis, and peers exchange a smaller fraction of data called block. To simulate the failure rate, every peer (leecher or seed) makes a decision to sent altered blocks based on a random uniformly distributed function, without enabling any globus security option. The results are presented in Figures 4 and 5. First of all, in all cases the download completes with an acceptable overhead, in contrast to GridFTP which has no mechanism of protection against these kinds of errors. Furthermore, we notice that as the failure rate increases transfers with smaller block sizes are more heavily affected, because one bad block causes the retransmission of all the blocks in a certain piece. So in cases where block size is $\frac{1}{8}$ of the piece size, the slowdown is 3 to 4 times in comparison with the case of a block the size of a piece and in failure rates up to 16%.

5.2 GridTorrent vs. GridFTP Performance

In this experiment we compare the performance of the GridTorrent prototype against the current GridFTP implementation in both Local and Wide Area Network environments. Specifically, we increase the number of concurrent requests over a single 128MB file from different physical nodes. Results for different file sizes (up to 512MB) are qualitatively similar. We measure the minimum, maximum and average completion time of this operation on all requesters. Our setup assumes a single server that seeds this file and up to 32 physical machines that issue simultaneous download requests. For the LAN experiments, we use our

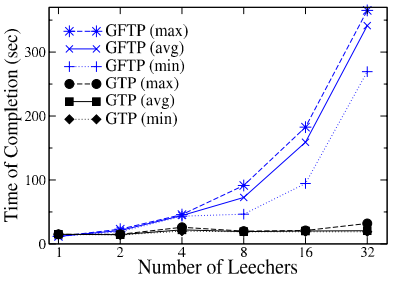

Fig. 6. Min, max and average time of completion for both GFTP and GTP over various number of downloaders in the LAN setting

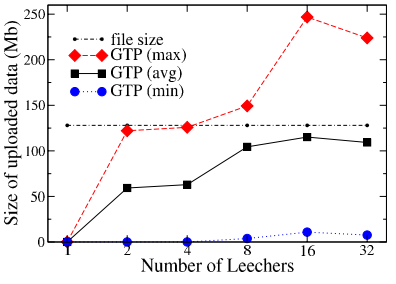

Fig. 7. Min, max and average size of uploaded data from leechers only in the LAN setting

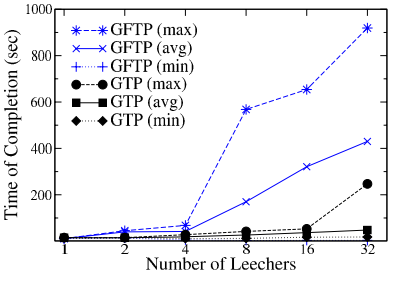

Fig. 8. Min, max and average time of completion for both GFTP and GTP over various number of downloaders in the WAN setting

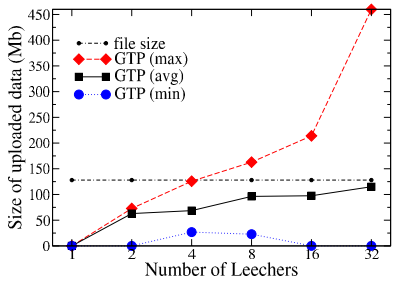

Fig. 9. Min, max and average size of uploaded data from leechers only in the WAN setting

laboratory cluster infrastructure with gigabit ethernet interconnect. For the WAN experiments, we allocate the same amount of nodes in PlanetLab [20, 6]. In this environment, there exist several heavily loaded nodes, geographically distributed with various network latencies and bandwidth constraints. Obviously, PlanetLab offers an environment more similar to a real world Grid environment, where requests may occur from different places over the globe using personal computers. Location information on the file, the list of peers that obtain or are currently downloading the file, as well as other file metadata are stored in DRLS, located in a single machine which simulates 30 nodes in a XOROS DHT.

In Figure 6, we present the completion times for the LAN setup. We notice that GridTorrent can be over 10 times faster than GridFTP in all measured times. This occurs for the largest number of leechers. GridFTP cannot enforce cooperation among nodes; thus a single server must accommodate all clients in an serialized manner. One would expect that GridFTP would not be affected by the flash crowd effect in a LAN, especially with the Gigabit ethernet connectivity, but this is not the case. GridTorrent shows remarkable performance in all

Table 2. The effect of the κ parameter in DRLS

κ	α	ϵ	Average Messages	Mean latency (sec)
20	3	2	44	1.37
15	3	2	42	1.06
10	3	2	30	0.83
5	3	2	22	0.61

three metrics, as they remain unaltered by the increase in requests. Our method can be readily employed to sustain flash crowd effects as, due to the increasing cooperation among peers, it effectively reduces the load of the single server and provides adaptive portions of the file to the rest of the nodes. In Figure 7, we present this cooperation in terms of bytes sent exclusively among the leechers. We notice that, as the number of leechers increase, this traffic increases, showing the clients' active part in this process. On average, each leecher seems to be responsible for sending almost one file's worth of data to the other leechers and no more than two times the file size in maximum.

Figure 8 summarizes our results from the WAN setup. It is evident that GridFTP cannot cope with increasing transfer loads in a real world environment. GridFTP's minimum times remain constantly low and close to GridTorrent's due to the fact that there always exists at least one leecher close to the single server that downloads the file faster. Furthermore, we register a major difference in GridFTP's maximum, minimum and average times (e.g., for 32 leechers the last one receives the file 30 times slower than the faster one and about 2 times slower on average). This large variance is due to the protocol's inability to cope with heterogeneity – small number of close nodes finish early while the rest of the clients that are not close to the server are drastically affected.

In GridTorrent, the closest nodes that finish faster are exploited and upload data to the remaining ones, decreasing the overall completion time that gracefully scales with the number of simultaneous leechers. Our method is 3 to 10 times faster both on average and in the worst case, while it exhibits very small variation between the three reported metrics. In Figure 9, we can see the level of cooperation between the leechers as they increase in numbers. We clearly notice a greater variance in the bytes sent by each peer compared to the LAN setting. This shows how adaptive GridTorrent is: Close nodes that finish early contribute to the other peers more than average, while the are few nodes that finish late and cannot share interesting data with the rest of the peers. The WAN experiment depicts in the best way why our protocol is a robust, bandwidth efficient means of file transfer that vastly outperforms current practices.

5.3 DRLS Performance

To evaluate the DRLS implementation we created a scenario where 64 peers, storing about 1000 items perform random lookups and updates at increasing rates. Measuring the mean number of messages and time required for each

operation reveals that in the absence of node churn, results remain almost constant, even when constantly doubling the request rate from 1 operation every 6 seconds up to 10 $\frac{operations}{sec}$. This suggests that the underlying XOROS protocol scales to flash-crowd usage patterns as expected. When participants start to leave and new ones enter some messages get lost, so nodes have to wait for timeouts to expire before proceeding with a command. Nevertheless, as the node population settles and routing tables are updated, the performance characteristics return to the expected levels. An interesting find is that during periods of churn, a higher request rate may result in more messages, but this helps nodes react quicker to overlay changes and refresh their routing tables faster.

During this series of experiments we have also investigated the impact of the various DHT parameters: κ, which controls the number of replicas kept for each data item and sets the quorum size for the mutex protocol, α, which defines how many parallel messages can be in-flight during an operation and ϵ, which marks the number of peers that may exhibit arbitrary behavior or fail before a request is completed. As expected, the replication factor κ plays the most important role in shaping both message count and latency. Table 2 summarizes the results for multiple runs of the aforementioned scenario, with different values of κ. Lowering κ reduces the number of nodes that should be contacted in each operation, thus causing the overall latency to drop. However, mean latency is not directly proportional to the number of messages, as a lot of communication is done in parallel. Dividing the latency numbers with the average messaging cost of 80 msec results in the mean number of messages have to be sent in serial order, either due to the protocol or the α parameter. When the network is small, like the case of 64 nodes, we believe that a replication factor of 5 should be enough. On the other hand, when deploying DRLS to a massive number of participants (i.e. a "Desktop Grid"), keeping κ to the default value of 20 can help avoid data loss in case of sudden network blackouts or other unplanned and unadvertised peer problems, even if the messaging cost is higher.

6 Conclusion

In this paper, we describe a P2P-based data management architecture that comprises of GridTorrent and DRLS. GridTorrent is a cooperative data transfer mechanism that maximizes performance by adaptively choosing where each node retrieves file segments from. DRLS is a distributed replica service which is based on a modified kademlia DHT, allowing efficient processing even during node churn. Our proposed solution is compatible with the current Data Grid architecture and can be utilized without any changes by already deployed middleware. Experiments conducted both in LAN and WAN environments (the PlanetLab infrastructure), show that GridTorrent vastly outperforms GridFTP, being up to 10 times faster. Moreover, experiments on DRLS in a dynamic environment show that the benefits of a peer-to-peer network can be readily exploited to provide a scalable Grid service without significant loss in performance. DRLS is able to provide reliable location services even when the load rates multiply.

References

1. The Large Hadron Collider, http://lhc.web.cern.ch/lhc/
2. The GREDIA Project, http://www.gredia.eu/
3. The official site of Globus Toolkit, http://globus.org/toolkit
4. The official BitTorrent client, http://www.bittorrent.org
5. Distributed.net, RSA Labs 64bit RC5 Encryption Challenge, http://www.distributed.net
6. PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services, http://www.planet-lab.org/
7. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S.: Data management and transfer in high-performance computational grid environments. Parallel Computing 28(5), 749–771 (2002)
8. Allcock, W., Bresnahan, J., Kettimithu, R., Link, M., Dumitresku, C., Raicu, I., Foster, I.: The globus striped gridftp framework and server. In: Proceedings of the ACM/IEEE Conference on Supercomputing, SC 2005 (2005)
9. Anderson, D.: Boinc: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (2004)
10. Cai, M., Chervenak, A., Frank, M.: A peer-to-peer replica location service based on a distributed hash table. In: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Pittsburgh, PA (November 2004)
11. Chazapis, A., Koziris, N.: Xoros: A mutable distributed hash table. In: Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2007), Vienna, Austria (2007)
12. Chazapis, A., Zissimos, A., Koziris, N.: A peer-to-peer replica management service for high-throughput grids. In: Proceedings of the 2005 International Conference on Parallel Processing (ICPP 2005), Oslo, Norway (2005)
13. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications (2000)
14. Chervenak, A., Palavalli, N., Bharathi, S., Kesselman, C., Schwartzkopf, R., Stockinger, H., Tierney, B.: Performance and Scalability of a replica location service. In: Proc. of the 13th IEEE International Symposioum on High Performance Distributed Computing Conference (HPDC), Honolulu (June 2004)
15. Cohen, B.: Incentives build robustness in bittorrent. In: Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA (June 2003)
16. Sullivan III, W.T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., Anderson, D.: New major seti project based on project serendip data and 100,000 personal computers. In: Astronomical and Biochem ical Origins and the Search for Life in the Universe, Proc. of the Fifth Intl. Conf. on Bioastronomy (1997)
17. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: Proceedings of the 5th ACM conference on Computer and communications security, pp. 83–92. ACM Press, New York (1998)
18. Kaplan, A., Fox, G., von Laszewski, G.: Gridtorrent framework: A high-performance data transfer and data sharing framework for scientific computing. In: Proceedings of GCE 2007, Reno, Nevada (2007)
19. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 53. Springer, Heidelberg (2002)

20. Peterson, L., Anderson, T., Culler, D., Roscoe, T.: A blueprint for introducing disruptive technology into the internet. In: Proceedings of HotNets–I, Princeton, NJ (October 2002)
21. Ripeanu, M., Foster, I.: A decentralized, adaptive, replica location service. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), Edinburgh, UK (July 2002)
22. Sherwood, R., Braud, R., Bhattacharjee, B.: Slurpie: A cooperative bulk data transfer protocol. In: Proceedings of IEEE INFOCOM (March 2004)
23. Stockinger, H., Samar, A., Holtman, K., Allcock, B., Foster, I., Tierney, B.: File and object replication in data grids. Cluster Computing 5(3), 305–314 (2002)
24. Thain, D., Basney, J., Son, S.-C., Livny, M.: The kangaroo approach to data movement on the grid. In: Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing, HPDC10 (2001)
25. Wei, B., Fedak, G., Cappello, F.: Collaborative data distribution with bittorrent for computational desktop grids. In: Proceedings of the 4th International Symposium on Parallel and Distributed Computing, ISPDC 2005 (2005)
26. Weigle, E., Chien, A.A.: The composite endpoint protocol (cep): Scalable endpoints for terabit flows. In: Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, CCGrid 2005 (2005)
27. Zissimos, A., Doka, K., Chazapis, A., Koziris, N.: Gridtorrent: Optimizing data transfers in the grid with collaborative sharing. In: Proceedings of the 11th Panhellenic Conference on Informatics, Patras, Greece (2007)