GridTorrent: Optimizing data transfers in the Grid with collaborative sharing

Antonis Zissimos, Katerina Doka, Antony Chazapis and Nectarios Koziris

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

Abstract

As Grid systems expand and become more and more popular, there is a growing need for efficient, scalable and robust data transfer mechanisms that can deal effectively with large file transfers and flash crowd situations. In this paper, we address the problem of data transfer optimization by presenting GridTorrent - a modified BitTorrent protocol, tightly coupled with modern Grid middleware components. GridTorrent can be used to transfer files directly from established GridFTP servers or other GridTorrent peers that are simultaneously requesting the same information. The peer-to-peer approach enables the aggregate data transfer throughput to escalate, even when numerous requests rely on a single data source, and achieve better utilization of the available Grid resources. Experimental results, conducted using a prototype implementation, suggest that there are significant advantages when using GridTorrent to optimize data transfers. Moreover, GridTorrent is completely backwards-compatible with already deployed Grids.

Keywords: Grid, Data Grid, data management, data transfer, replica location service, replica optimization service, peer-to-peer networks, collaborative sharing

1. Introduction

In recent years, Grid systems have gained popularity and have been widely utilized within the scientific community. The Grid is a wide-area, large-scale distributed computing system, in which remotely located, disjoint and diverse processing and data storage facilities are integrated under a common service-oriented software architecture [Foster, Kesselman (1999)]. One of the Grid's most essential and critical components is the data management layer. Modern Grid architectures employ several services for this fundamental layer. A basic service is the Data Transfer service, responsible for moving files among grid nodes (e.g. GridFTP). Also, the Replica Location Service (RLS) keeps track of the physical locations of files. Moreover, an optimization service 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.

In this paper we introduce GridTorrent, which is focused in realtime optimization of data transfers on the Grid. GridTorrent is an implementation of BitTorrent designed to interface and exploit well-defined and deployed Data Grid components and protocols. We argue that a protocol based on peer-to-peer techniques can provide the needed effectiveness and scalability, even under extreme load and flash crowd conditions. We ground our work on BitTorrent due to its sustained throughput compared to other peer-to-peer data transfer protocols under the aforementioned scenarios.

GridTorrent can be integrated into existing middleware distributions offering data transfer optimization without requiring extra services from the Grid infrastructure. For instance, instead of *.torrent* metainfo files, GridTorrent uses the RLS provided by existing Grid deployments. Also, the GridTorrent client is able to use GridFTP and its partial file transfer feature to request file fragments from servers that already hold a file. Therefore, a GridTorrent is possible to connect both to other GridTorrent clients and GridFTP servers located in the Grid. GridTorrent use the inherent mechanism of the BitTorrent protocol that selects the best download sources on the fly, based on realtime bandwidth metrics, independent of the actual transfer mechanism. In addition, GridTorrent deployments are backwards compatible, because additional data stored in the Replica Location Service is ignored from regular clients of the Data Management API and considered only from GridTorrent enabled clients.

The rest of the paper is organized as follows: Section 2 presents an overview of Data Management Services in the Grid. The BitTorrent protocol is briefly described in Section 3. Our prototype is presented in section 4 and evaluated through performance results of its implementation in section 5. Finally we conclude the paper with references to related work in the area and thoughts on future work in the same direction.

2. Data management services in the Grid

2.1 Locating files

One of the core building blocks of the Data Grid architecture is the Replica Location Service. The Grid environment may require that data is to be scattered globally due to individual site storage limits, but also remain equally accessible from all participating computing elements. In such cases, it is common to use local caching of data to reduce the network latencies that would normally add up as a constant overhead of remote data access operations. In Grid terminology, local copies of read-only remote files on storage elements are called *replicas* [Stockinger et. Al. (2002)], while applications running on the Grid request such local file instances through specialized Grid data management services. To work with a file, a Grid application must first ask the Replica Location Service to locate corresponding instances of the requested item so that if a local replica already exists, the application can use normal file semantics to access its contents. In the case only remote copies are found, another component of the Data Grid can take on the responsibility of copying the remote data to the local node and update the replica location indices with the position of the new instance.

The most widespread solution currently deployed on the Grid, namely the Giggle Framework [Chervenak et Al. (2004)], constructs a uniform filename namespace of unique per VO identifiers (logical filenames - LFNs) and manages the mappings of these identifiers to physical locations of files (physical filenames - PFNs). LFNs are used by the applications to locate data, no matter the source of the request or the physical location of the information. PFNs, which are used by the Replica Location Service and other Data Grid services, are structured similar to a URL, describing the access protocol, the site and the path in the site directory structure for a given replica. In order to distribute the replica location data throughout the Grid, Giggle makes use of two main components, the local replica catalogs (LRCs) and the replica location indices (RLIs):

- An LRC maintains information about logical filenames such as access lists, creation date and various other file attributes. It also stores a map of all physical filenames that are replicas of a logical filename (LFN to PFN maps). Given an LFN, the LRC will return the associated PFN set.
- An RLI maintains information about the catalogs and the associated logical filenames. It can find which catalog holds the replica file list for a given LFN (LFN to LRC map).
- In a default deployment scenario, each participant of a VO manages an LRC, while the overall orchestration of the RLS is done by a single central RLI per VO. When requirements escalate, multiple RLIs can be deployed in parallel, providing optional coarse-grain load-balancing and fail-over features to the replica location infrastructure. The Giggle Framework instructs that multiple indices and catalogs form a two-level hierarchy, with each LRC linked to multiple RLIs and vice versa. Multiple RLIs can also form tree-like structures.

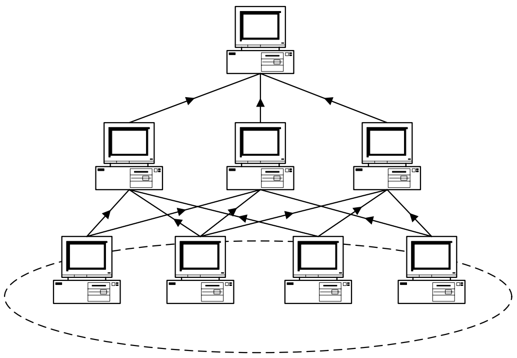

Figure 1. *Replica Location Service deployment scenario.*

2.2 Transferring files

Another fundamental building block of the Data Grid architecture is the GridFTP Protocol, a protocol defined by the Global Grid Forum, that addresses the issue of data transfer among grid nodes. Modern Grid middleware distributions like the Globus Toolkit [Foster et Al. (2001)] include the GridFTP service, as it has become an integral part of the Grid infrastructure. It extends the standard FTP protocol by introducing features like Grid Security Infrastructure (GSI) [Foster et. Al. (1998)] and third-party control of data transfer. The later provides the ability to a user or an application at one site to initiate, monitor and control a data transfer operation between two other "parties", the source and destination sites for the data transfer. This is achieved through the separation of control and data channel. An additional extension is the support of both manual setting and automatic negotiation of TCP buffer sizes so as to optimize the transfer of large files and large sets of small files.

However, the techniques adopted by the Grid community over the past years in this direction have evolved significantly. The most widespread solution currently deployed on the Grid, is the Globus Striped GridFTP protocol [Allcock et. Al. (2005)], included in the current release of the Globus Toolkit 4. The new features added to GridFTP enable transfer of data striped or interleaved across multiple servers, partial file transfer, meaning transfer file portions rather than the complete file and parallel data transfer using multiple TCP streams in parallel, not only between a single source and destination, but also between each of the multiple servers participating in a striped transfer.

Even so, the GridFTP protocol is based on the client – server model, inflicting all the undesirable characteristics of centralized techniques, such as server overload, the existence of a single point of failure and the inability to cope with flash crowds. However, the Replica Location Service can be exploited to optimize the data movement services. Through replica-aware algorithms, data movement services can take advantage of multiple replicas to boost aggregate transfer throughput. We argue that the centralized approach used may reach its limits, when the number of potential downloaders and the volume of data increase in several orders of magnitude.

3. Brief description of the BitTorrent protocol

BitTorrent [Cohen (2003)] 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 download time for large, popular files and the load on servers that serve these files as well.

Every file is divided in chunks, typically of 256kB each. Clients can exploit this fragmentation by simultaneously downloading chunks from many sources. For integrity reasons imposed from this extended fragmentation, a hash is kept for every group of chunks called a piece. This information, along with the file size, is stored in a metainfo file, identified by the extension *.torrent*. At bootstrap, a client first reads the metainfo file to obtain the fragmentation policy and the URL of the tracker. The tracker service is responsible for holding information concerning the peers involved in a file transaction. Upon connection to the corresponding tracker, the client receives a random list of peers that are currently downloading or uploading the file. Peers are categorized in *seeds* when they already have the whole file and *leechers* when they are still downloading pieces. From that point on, the client is responsible to establish connections to those peers and decide all the data movements based on local information.

In order to improve availability and ensure uniform distribution of the pieces, a peer selects the pieces to be downloaded based on a *rarest-first policy*. The effectiveness of BitTorrent relies on its built-in incentive mechanism, the *choking algorithm*. It is essentially a peer selection algorithm that poses a limit on the number of concurrent uploads, typically set to 4, and gives priority to the peers with the best upload rates. There is a *rechoke period* after which each peer recalculates the upload rates of its neighbors and decides which peers to *choke* and which to *unchoke*. Thus, the protocol avoids *free-riders*, meaning peers that download without contributing to the system. Furthermore, an additional peer is randomly unchoked once every third rechoking period by means of an *optimistic unchoke*. A BitTorrent deployment scenario is depicted in Figure 2(a).

The latest version of the BitTorrent client replaces the tracker service with a Distributed Hash Table (DHT) for dynamically locating the peers that participate in a file transaction.

4. GridTorrent design

4.1 Extending the Replica Location Service

A main component in the BitTorrent protocol is the metainfo file, also known as the *.torrent* file. This file contains important information about the data to be downloaded. In our design, we eliminate the use of this file and incorporate all the necessary information into the Grid Catalogs and more specifically in the Replica Location Service. For each file we add the following attributes in the RLS catalog:

- the file size
- the size of each piece
- the hash of each piece (optional)

The information stored in the Replica Location Service defines the number of replicas for each file as well as the physical location of the actual data. The physical location is identified by a unique physical file name (PFN) such as a GridFTP URL. To enable the use of the GridTorrent protocol we introduce the GridTorrent URL, which has the form:

btp://site.fully.qualified.domain.name:port/path/to/file

gtp://site.fully.qualified.domain.name/path/to/file

One advantage of the proposed modification is the use of already implemented features to model our solution, preserving the backwards compatibility of the existing Grid Architecture. The extra data stored in the Replica Location Service is by default ignored by existing applications that cannot recognize it. If an application cannot translate the GridTorrent URL, it will simply ignore the corresponding replica. Therefore, the proposed changes in the current Grid Architecture not only enhance the performance of data transfers, but furthermore seamlessly integrate with the current state-of-the-art in Grid Data Management.

4.2 Extending the Data Transfer Service

GridTorrent is a Grid-enabled version of a BitTorrent client. It uses the existing Grid protocols to provide an optimized data transfer service, as it has the ability to directly communicate with GridFTP servers. The GridTorrent design comprises of the following components:

- The RLSManager, which handles all the communication with the Replica Location Service. This component is responsible for finding file information (file size, piece size), registering replicas in the RLS and finding existing replicas.
- The PeerManager, which handles all the communication with other GridTorrent or GridFTP enabled peers.
- The DiskManager, which handles all disk I/O for storing and retrieving files. If file hashing is enabled, the DiskManager is also responsible for verifying the correctness of the file by comparing the SHA1 of each downloaded piece against the SHA1 provided by the RLS.

The main advantage of GridTorrent is the inherent optimization algorithm for replica selection. A request to GridTorrent for a file, will trigger a query to the Replica Location Service. This query can occur periodically to be notified of any changes in the locations of file replicas or of any joins or departures of nodes, which can be GridFTP servers as well as GridTorrent leechers or seeds. Upon receiving the list of peers, GridTorrent acts according to the protocol prefix of the pfn. If it concerns a GridTorrent client, the two involved peers initiate communication by exchanging the bitTorrent *bitfield* message, informing each other of the pieces they possess.

Furthermore, each time a peer downloads a piece, it sends a *have* message notifying all peers connected to it of its new acquisition. In order to download data from another GridTorrent client, the peer issues a *request* message for blocks. Blocks are parts of a piece, referenced by the piece index, a zero-based byte offset within the piece and their length. Having information about the available pieces, GridTorrent starts downloading pieces in a random order. In case of a GridFTP server, the peer does not need to exchange *bitfield* messages. As for the downloading technique, the client issues a GridFTP partial get message for the data within the specific block it intends to download. The selection of the block is performed in the same way. A GridTorrent deployment scenario is depicted in Figure 2(b).

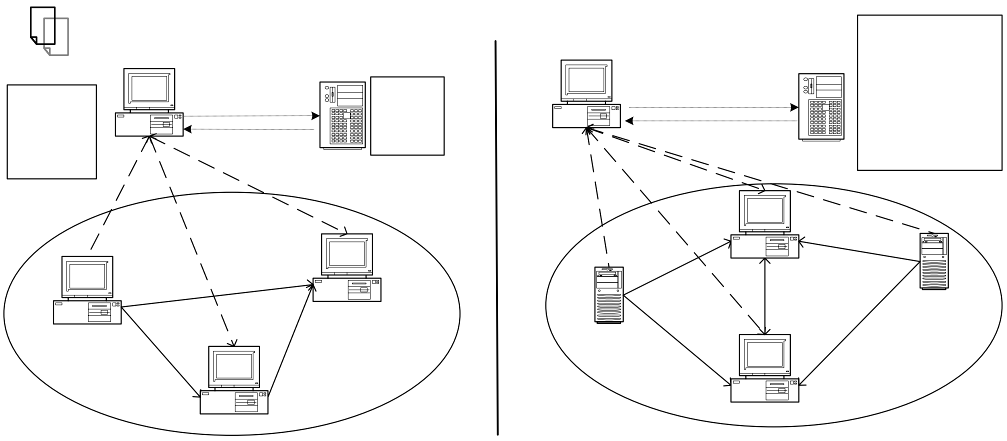

Figure 2. *(a) BitTorrent deployment scenario (b) GridTorrent deployment scenario*

One can exploit the information gained from the *bitfield* and *have* messages for applying a specific download strategy. An example is downloading pieces in rarest first order, as described earlier. One can also optimize the piece selection policy of the client, preferring the peers with better download rates. Whilst the pieces are downloading from the various peers, GridTorrent client maintains statistics about their mean transfer time. Each time a block is downloaded by a peer the mean transfer time of this node is calculated using the current download rate and the peer's history.

The GridTorrent protocol uses the BitTorrent tit-for-tat algorithm to ensure that all peers contribute to the file downloading and to discourage *free-riding*. Each peer has to upload to a constant number of N peers each time, utilizing the aforementioned optimistic unchoking mechanism.

5. Implementation and experimental results

Our GridTorrent prototype implementation is entirely written in Java. The

GridTorrent client has bindings with Globus Toolkit 4 libraries and exploits the GridFTP client API and the Replica Location Service API. For our experiments we use the PlanetLab infrastructure. Our testbed consists of 18 PlanetLab nodes, completely heterogeneous. One of them is dedicated to the RLS service, another is used as a GridFTP Server and the remaining nodes are file transfer clients.

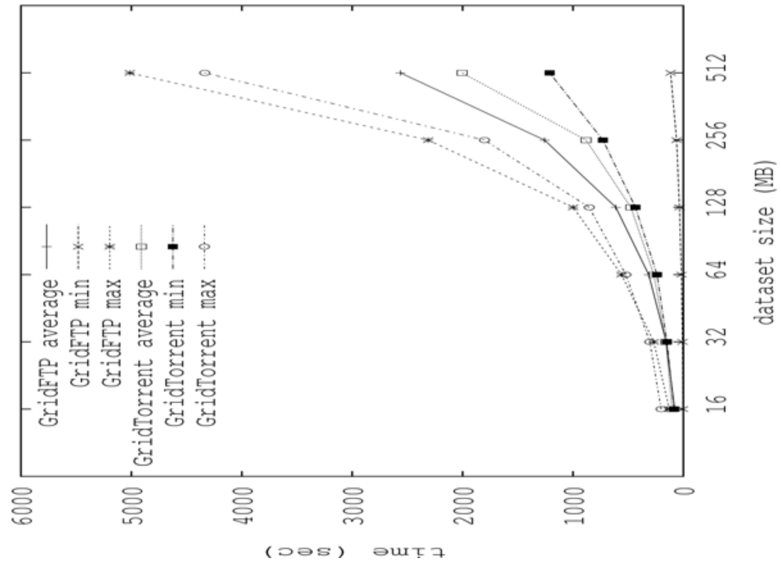

***Figure 3.** Minimum, maximum and average time required for 16 independent downloaders to transfer a file using either GridFTP or GridTorrent.*

Figure 3 summarizes experiment results of our prototype, presenting the minimum, maximum and average completion time in seconds. We compare the performance of GridTorrent versus GridFTP when distributing a file to a constant set of nodes. Having a GridFTP server running on a node, we invoke concurrent file transfers from the GridFTP server to 16 client nodes of our testbed, first using GridFTP and then using GridTorrent, and measure the total completion time. This experiment is conducted for file sizes varying from 16 MB to 512 MB.

The first conclusion derived from concentrating on the mean completion time curves, is that GridFTP performs better for small file sizes, while GridTorrent results in faster transfers of larger files. As observed, there is a crossover point at 32 MB beyond which GridTorrent clearly outperforms GridFTP. This is due to the overhead introduced by the inherent BitTorrent protocol and its fragmenting mechanism - a client must contact the RLS, obtain information about the file and receive the list of peers involved in the file transfer before contacting them. Moreover, the messages exchanged among GridTorrent clients, apart from the messages actually containing file chunks, constitute an overhead, which in the case of small file sizes cannot be compensated by the parallelism offered by the protocol.

Examining the figure more carefully, we conclude that GridTorrent's maximum and average completion time is in general better than that of GridFTP, whereas its minimum completion time is considerably higher. This can be justified by the fact that the GridFTP server and one of the participating clients are located in the same LAN. Since GridTorrent's protocol concentrates on fairness among nodes, it cannot take as much advantage of LAN speeds as GridFTP does. In general, in cases where bandwidth is not an issue, GridFTP can perform faster in contrast to cases where bandwidth is the bottleneck and GridTorrent's parallelism can boost performance.

6. Related work

The efficient movement of distributed volumes of data is a subject of constant research in the area of distributed systems. Having already analyzed the current data transfer practices in Grid environments, in this section we focus on other proposed data movement techniques, centralized or in the context of the peer-to-peer paradigm.

The Kangaroo architecture [Thain et. Al. (2001)] is a data transfer system that aims at better overall performance by making opportunistic use of a chain of servers. The disks are used as buffers, hiding network latencies. Thus it can be used in environments where high throughput is needed but consistency is not of major concern.

Weigle and Chien [Weigle, Chien (2005)] conceptualize the N-to-M communication problem (M readers, N writers) by proposing the Composite Endpoint Protocol (CEP) as a solution for bulk transfers. Users provide high-level transfer data to CEP and a scheduler generates a schedule which optimizes the transfer performance by producing a balanced weighting of a directed graph using various algorithms. Afterwards, readers and writers implement the provided schedule. Nevertheless, the model remains centralized.

Slurpie [Sherwood et. Al. (2004)] follows a similar approach to BitTorrent, as it targets bulk data transfer and makes analogous assumptions. Nonetheless, it does not support connection chocking, a feature of BitTorrent which encourages cooperation.

Several papers and technical reports analyzing BitTorrent's performance have been published. Some of them present the conclusions derived from BitTorrent tracker and client logs ([Izal et. Al. (2004)], [Pouwelse et. Al. (2004)]). Related work aims to estimate the global efficiency of BitTorrent, the client interactions and the scalability of the system under flash-crowd conditions [Izal et. Al. (2004)] and present a detailed measurement study of the behaviour of BitTorrent in terms of popularity, availability, download performance, content lifetime and pollution level [Pouwelse et. Al. (2004)]. In [Qiu, Srikant (2004)], the authors attempt to theoretically trace the file-sharing system's behaviour by introducing a simple fluid model and by obtaining expressions for the average number of seeds, the average number of downloaders and the average download time as functions of various system parameters. Moreover BitTorrent's built-in incentive mechanism is evaluated. A simulator approach to understand BitTorrent performance is presented in [Bharambe et. Al. (2005)]. The authors examine the potential unfairness of the protocol under certain circumstances. The evaluation shows that BitTorrent can treat unfairly high bandwidth peers. To solve this problem the authors suggest a block-level rather than a rate-level tit-for-tat policy.

[Wei et. Al (2005)] compares BitTorrent to FTP for data delivery in Grid environments. Their experiments conducted in a LAN cluster involving 20 and 64 nodes, demonstrate that BitTorrent is efficient for large file transfers and scalable when the number of nodes increases. Nevertheless, in contrast to [Wei et. Al (2005)] we focus on Data Grid environments with specific concern for flash-crowd situations, rather than Computational Desktop Grids. Moreover, we propose an architecture which can be directly deployed in a real-life Grid environment.

7. Conclusion

We believe that in future Data Grid deployments client-server data transfer mechanisms will be put aside in favour of peer-to-peer techniques, which provide better network traffic distribution and resolve the single point of failure problem. In this paper we described such a peer-to-peer data transfer algorithm, namely GridTorrent, based on the popular BitTorrent protocol. GridTorrent is compatible with the current Data Grid architecture and can be utilized without any changes in already deployed Grid middleware.

Experiments, conducted in the PlanetLab infrastructure, show that GridTorrent outperforms GridFTP in cases of large data transfers or limited bandwidth between nodes. On the contrary, GridFTP performs better when there are no bandwidth constraints. However, even in such cases, using GridTorrent to transfer data enables remote peers to exploit the collaborative sharing properties of the underlying BitTorrent protocol, in order to boost aggregate performance.

In future work, we plan to further evaluate our implementation using a larger set of possible data transfer scenarios and different protocol parameters.

References

Allcock, W., Bresnahan, J., Kettimithu, R., Link, M., Dumitresku, C., Raicu, I., Foster, I. (2005), *The Globus Striped GridFTP Framework and Server*, in Proc. of the ACM/IEEE Conference on Supercomputing, SC'05.

Bharambe, A., Herley, C., Padmanabhan, V. (2005), *Analyzing and Improving BitTorrent Performance*, Technical Report, Carnegie Mellon Institute and Microsoft Research.

Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunszt, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K., Tierney, B. (2002), *Giggle: a framework for constructing scalable replica location services*, In Proc. of the 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland.

Cohen, B. (2003), *Incentives Build Robustness in BitTorrent*, Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA.

Foster, I., Kesselman, C., Tuecke, S. (2001), *The Anatomy of the Grid: Enabling Scalable Virtual Organizations* International Journal of Supercomputer Applications, vol. 15.

Foster, I., Kesselman, C., Tsudik, G., Tuecke, S. (1999), *A security architecture for computational grids*, of the 5th ACM conference on Computer and communications security, San Francisco, California, United States.

Izal, M. ,Uroy-Keller,G., Biersack, E. W., Felber, P. A., Al Hamra, A., Garces-Erice , L. (2004), *Dissecting BitTorrent: Five Months In Torrent's Lifetime,* in Proc. of the 5th Passive and Active Measurement Workshop.

Pouwelse, J. A., Garbacki, P., Epema, D., Sips, H. J. (2004), *A Measurement Study of the BitTorrent Peer-to-Peer File Sharing System*, Technical Report, Delft University of Technology..

Qiu, D., Srikant, R. (2004), *Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks*, in Proc. of SIGCOMM Comput. Comun., vol. 34, pp. 367-378.

Sherwood, R., Braud, R., Bhattacharjee, B. (2004), *Slurpie: A Cooperative Bulk Data Transfer Protocol*, in Proc. IEEE INFOCOM.

Stockinger, H., Samar, A., Holtman, K., Allcock, B., Foster, I., Tierney, T. (2002), *File and Object Replication in Data Grids*, Cluster Computing, Kluwer Academic Publishers, vol. 5, pp. 305-314.

Thain, D., Basney, J., Son, S. C., Livny, M. (2001), *The Kangaroo Approach to Data Movement on the Grid*, in Proc. of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10).

Wei, B., Fedak, G., Cappello, F. (2005), *Collaborative Data Distribution with BitTorrent for Computational Desktop Grids*, in Proc. of the 4th International Symposium on Parallel and Distributed Computing, ISPDC'05.

Weigle, E., Chien, A. (2005), *The Composite Endpoint Protocol (CEP): Scalable Endpoints for Terabit Flows,* in Proc. of the IEEE International Symposium on Cluster Computing and the Grid, CCGrid'05.