P2P File Sharing for P2P Computing

Cyril Briquet[1], Xavier Dalem,
Sébastien Jodogne , Pierre-Arnoul de Marneffe
EECS Department, University of Liège, Belgium

Abstract

The transfer of large input data files in P2P computing Grids often leads to delays in Task completion times. Existing research related to this topic has been focused on the spatial grouping of Tasks, i.e. reuse of available data through data caching and data-aware scheduling. However, it tends to decrease the level of parallelism of Task execution. In this paper, this issue is addressed by integrating the BitTorrent P2P file sharing protocol, a novel Task selection scheduling algorithm, an existing online, data-aware Resource selection algorithm (similar to Storage Affinity), and caching support. These algorithms have been implemented in the Lightweight Bartering Grid middleware. The Java implementation relies exclusively on Free and Open Source data transfer software (Azureus, Apache FTP server, edtFTPj). The proposed data transfer architecture does not need Predictive Communications Ordering or an explicit deployment of an overlay network. It is also easily deployable. Our main contribution is the joint use of P2P computing and P2P file sharing technologies, enabling a highly scalable and adaptive data transfer architecture to support P2P computing.

Keywords Grid, Peer-to-Peer, P2P, BitTorrent, file sharing, caching, Bag of Tasks, scheduling

[1]Corresponding author: Cyril Briquet, Department of Electrical Engineering and Computer Science, University of Liège, Montefiore Institute, B37, B-4000 Liège, Belgium
E-mail: briquet@montefiore.ulg.ac.be Telephone: +32 4 366 26 09 Fax: +32 4 366 28 74

1 Introduction

Peer-to-Peer (P2P) Grid computing, which seeks the convergence of Grid and Peer-to-Peer technologies, is defined as computational resource sharing in Grids organized into P2P networks. A P2P Grid is defined as a transient Virtual Organization that emerges in a bottom-up fashion, out of exchanges of computing time between Peers connected through a P2P network. Resources providing computing time are managed by Peers. As P2P networks operate at the edge of the Internet, Resources are typically edge computers, i.e. unmanaged, not necessarily fast or reliable processing units. A consumer Peer is a Peer which consumes computing time of Resources from other Peers. A supplier Peer is a Peer which supplies the computing time of its Resources to other Peers.

A Bag of Tasks (BoT) is a set of independent computational Tasks, that may process identical input data files. A Data-Intensive BoT [19, 29, 30] is a Bag of Tasks that processes large input data files. User Agents, which are submission interfaces to the Grid, submit BoTs to Peers. In turn, Peers schedule BoTs to their Resources or other Peers' Resources. Scheduling Data-Intensive BoTs is not trivial in a P2P Grid because transfers of large data files have an impact both on the overall completion times of the Tasks and on the willingness of Peers to actually supply computing time. Indeed, a Peer may lose patience while waiting for its turn to download input data files required by external Tasks it has accepted to compute. Data transfers should therefore be taken into account to prevent multiple simultaneous transfers of identical large data files from creating performance bottlenecks. Furthermore, to achieve low completion times, Task execution parallelism should be maintained rather than avoided.

We propose to maximize both data reuse and parallelism of Task execution by combining (1) the BitTorrent P2P file sharing protocol to transfer input data files, (2) a Grid-wide deployment of data caching support, (3) data-aware Peer and Resource selection scheduling algorithms (similarly to Storage Affinity [30]), and (4) a novel Task selection scheduling algorithm, called Temporal Tasks Grouping (TTG). A flash crowd is a large number of downloaders simultaneously downloading input data files. TTG is based on the concurrent scheduling of Tasks depending on identical input data files. It generates so-called flash crowds in a controlled way in order to synchronize data transfers on Tasks schedule so as to benefit from the full performance of BitTorrent.

This paper, which is an extended version of recent work [9], is structured as follows. The proposed data transfer architecture and the Task scheduling algorithms are introduced in Sections 2 and 3, respectively. Experimental results are presented in Section 4 and discussed in Section 5. Related work and state of the art in the domain are reviewed in Section 6. Finally, Section 7 concludes the paper.

2 Data Transfer Architecture

Our proposed data transfer architecture enables the transfer of input data files with either BitTorrent or FTP. It also takes into account the nature of P2P Grids and of the data transfer protocols themselves. It is also highly scalable.

In this paper, the context involves both P2P Grids and P2P file sharing networks. Consequently, in the following the term Peer, when designating a BitTorrent Peer, will be systematically substituted with the term BitTorrent Node. The term Peer will also be substituted with the term Grid Peer, except if it is already prefixed, i.e. consumer Peer or supplier Peer. This disambiguates the usage of the term Peer.

2.1 Data Preprocessing

When a BoT is submitted to a Grid Peer by a User Agent, all input data files to be processed by Tasks of a BoT are transferred with the BoT. Before the Grid Peer queues the BoT, it preprocesses the embedded input data files. These files are stored into the Grid Peer data cache, and stripped from the Tasks as necessary. Metadata are created for each input data file, then embedded into the BoT. The metadata of an input data file [18] consist of a Grid-level file name, a digital fingerprint of the file (i.e. a SHA-256-generated hash) and some location data. As the embedded input data files are removed and summarized by their metadata, Tasks themselves can be quickly transferred to multiple Resources or other Grid Peers.

A simple file naming scheme is used to provide Grid-wide naming unicity, e.g. PeerName.UserName.FileName. Location data is either an FTP URL or BitTorrent torrent metadata generated with the automated creation of a torrent out of the file.

As metadata generation can take some time for large files, the operation of submitting Tasks (i.e. transferring Tasks from a User Agent to a Peer) will inherently be faster if input data files are already cached (data caches will be presented in Section 2.5) because their metadata will already be computed.

2.2 Data Scheduling

The data transfer mechanism is the following: Input data files of a given Task are transferred synchronously upon scheduling of the Task to a given Resource. When a Task is scheduled to a Resource, this triggers the Resource to initiate the downloading of the required, uncached input data files. It is the Resource that actually initiates and controls the file download, similarly to what is proposed in the Super-Peer model [16]. The data transfer mechanism is thus pull-based.

The timing of data transfers, or data scheduling, is automatically synchronized with the timing of Tasks scheduling, which is explained in Section 3.

2.3 Data Paths

2.3.1 Data Transfer Endpoints

A Grid Peer may schedule its Tasks to its own Resources, or to other Grid Peers. A Grid Peer may also schedule external Tasks from other Grid Peers to its own Resources. Endpoints of a data transfer in a P2P Grid vary according to the type of Task for which it is required.

A data transfer is initiated similarly for local Tasks and external Tasks. It is the Resource where the Task is scheduled that initiates, and constitutes the sink of, the data transfer. The source of a data transfer for a local Task is the owner Peer of the Resource running the Task, which acts as both consumer and supplier Peer. The source of a data transfer for an external Task is its consumer Peer, not the supplier Peer that owns the Resource running the Task.

A consumer Peer consumes computing time to other Grid Peers and it can be remarked that it *"supplies"* its input data files to Resources. Input data files are not transferred immediately upon submission of a local Task to a supplier Peer. Data transfers take place only after the supplier Peer has scheduled this Task - perceived as an external Task - to one of its Resources. Moreover, such transfers never involve the supplier Peer and are strictly between the consumer Peer and supplied Resources.

2.3.2 Path of Data Transferred with FTP

If a Resource uses the FTP data transfer protocol to download a given input data file, this file is directly downloaded from the Grid Peer that shares it.

2.3.3 Path of Data Transferred with BitTorrent

If a Resource uses the BitTorrent data transfer protocol to download a given input data file, this file is simultaneously downloaded from, and shared with, other Grid Peers and Resources already sharing or downloading it. The data path of a given file in the Grid overlay, from the Grid Peer that shares it to the Resource that downloads it, may therefore not be direct. Moreover, a Resource may download an input data file using BitTorrent entirely from other Resources which have already downloaded it, and not at all from the Grid Peer that shares it.

2.3.4 Path of Metadata

As explained in Section 2.1, metadata describing an input data file required by a Task are actually embedded within the Task, and thus transferred to a Resource along with the Task. In particular, the BitTorrent torrent metadata of a given input data file - which is used to locate the BitTorrent tracker for the given file, as explained in Section 6.2 - are communicated in this way. The path and transfer timing of metadata are thus identical to those of the Tasks they are embedded into.

2.3.5 Summary of Possible Data Paths

Figure 1 shows the multiple data paths of input data files and, implicitly, of metadata in the Grid overlay. Plain lines represent the path of a Task from a consumer Peer to a supplier Peer, and then to the Resource where it is actually scheduled. Plain lines thus also represent the path of metadata, which are embedded within Tasks. It should be noted that when a Grid Peer schedules the Task directly to one of its own Resources, the representations of the consumer Peer and supplier Peer should be collapsed on Figure 1. Long-dashed lines represent the data path of FTP-transferred input data files. Short-dashed lines represent the data path of BitTorrent-transferred input data files. Resources in the cloud of Resources represented at the bottom of Figure 1 may belong either to the consumer Peer, to the supplier Peer, or to other Peers that are not involved in the represented Resource exchange but that have previously acted as supplier Peers to the represented consumer Peer.

2.4 Data Managers

A software component called Data Manager equips each Grid Peer and each Resource. It manages data transfers (sharing, download) and data storage. A Data

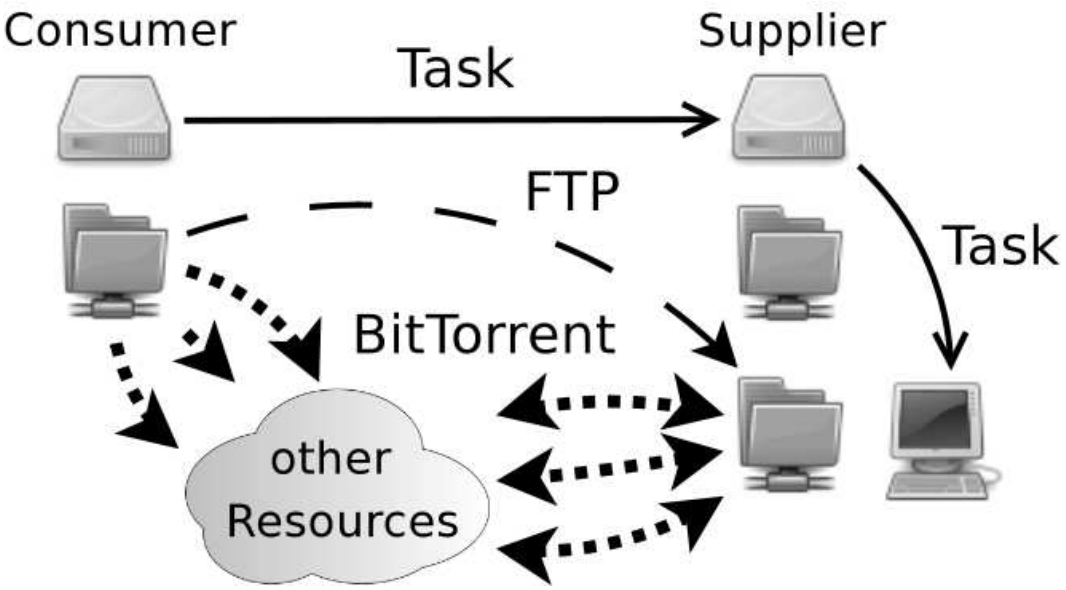

Figure 1: Possible data paths for the transfer of a single file from a consumer Peer (top left) to a supplier Peer (top right) and finally to a Resource of the latter.

Manager provides support for (1) data storage, (2) BitTorrent data sharing, tracking and downloading, and (3) FTP data sharing and downloading.

A Data Manager running on a Grid Peer manages input data files of its own Tasks only. The Peer's Data Manager is used to download input data files from its User Agents, and to store and share them with Resources processing Tasks depending on them.

A Data Manager running on a Resource manages input data files of Tasks belonging to its owner Peer and possibly to any consumer Peer of its owner Peer. The Resource's Data Manager is used to download input data files from its owner Peer or from other Resources, and to store and share them with other Resources processing Tasks depending on them.

Responsibilities of a Data Manager running on a Grid Peer and on a Resource therefore overlap but are not equal.

Each Data Manager is equipped with FTP and BitTorrent software components to manage data transfers. It is also equipped with one data cache software component, in order to manage data storage. Each Data Manager running on a Grid Peer runs a BitTorrent tracker, a BitTorrent client, and an FTP server. Each Data Manager running on a Resource runs a BitTorrent client and an FTP client.

All the data transfer software, both clients and servers, are embedded within the Data Managers, not requiring any extra support.

2.5 Data Caches

Each Data Manager is equipped with a data cache that manages the storage of data files. A Peer data cache is very basic, while more requirements are imposed on a Resource data cache.

A data cache running on a Grid Peer has an unlimited file capacity because it stores input data files of its own Tasks and a Grid Peer is supposed to be able to accept Tasks submitted by its User Agents. Of course, some limit could be imposed as a practical measure.

The available storage capacity on Resources is considered to be finite, as opposed to related works [30]. A data cache running on a Resource has a finite file capacity because Resources in a P2P Grid are typically edge computers.

2.5.1 Data Cache Parameters

The only supported operation on a data cache running on a Resource is the synchronization with a set of files, called working set, communicated by the Grid Peer that owns the Resource. A Resource's data cache is controlled by three parameters: (1) a cache size, (2) a working set and (3) a cache replacement policy.

2.5.2 Cache Size

The cache size bounds the maximum number of files that can be stored and is configured statically by the human administrator of the Resource.

2.5.3 Working Set

Each Grid Peer maintains a separate working set for each of its Resources. The working set is the set of files that the data cache of the Resource must have in storage. It is communicated by the Grid Peer each time a Task is scheduled to the Resource (see Section 2.2), and may also be communicated asynchronously. Each item in the working set is the metadata of an input data file (see Section 2.1).

Upon reception of a working set, a data cache synchronizes its contents with items described in the working set. It first verifies which items are already cached. The verification that a file belonging to the working set is already stored in a data cache is based on some of the metadata computed by the consumer Peer at submission time, i.e. Grid-level file name and hash.

When a file in the working set is not stored in the data cache, the Resource downloads it from the Grid Peer sharing it (which is either its owner Grid Peer or the consumer Peer, depending upon the origin of the Task using this file). As already mentioned, when BitTorrent is used, the Resource may also very well download it from other Grid nodes.

Each Grid Peer guarantees a property for each of the working set it maintains for its Resources: The working set maintained for a given Resource always includes the files needed by the Task running on, or scheduled to, the Resource. Each Grid Peer also guarantees that the working set defined for a Resource can always be fully stored in the Resource data cache. Consequently, a Grid Peer never schedules a Task to a Resource with insufficient cache size.

2.5.4 Cache Replacement Policy

Files are first downloaded before being inserted into the cache. The cache replacement policy selects which files to eject from the cache when the insertion of files from the working set causes an overflow (e.g. when the number of files exceeds the cache size). Files not part of the working set are ejected from the cache following a Least Recently Used (LRU) policy until the working set is fully stored. As a consequence of the cache replacement policy, an input data file cannot be ejected from a Resource data cache as long as it is needed by the currently running Task.

A BitTorrent network for a given file at a given moment in time is defined as the network of BitTorrent Nodes that store one or more pieces of this file. As long as a file is stored in a data cache, it remains shared with BitTorrent. BitTorrent networks are intrinsically transient. In the context of P2P Grids, the size of BitTorrent networks is variable and depends upon the contents of Resource data caches. On a given Grid Peer, as long as there is at least one BoT yet to be completed that depends on a given input data file, the size of the BitTorrent network for this file will be at least one, as the Grid Peer will continue to share this file until it is not needed by any Task of its queued BoT.

2.6 Data Trackers

Each Grid Peer is equipped with a software component called Data Tracker. It is not related to the BitTorrent tracker although it derives its name from a related purpose. It has the responsibility to continuously track the contents of the data caches of its Resources. It is actually a reverse mapping of input data files to the Resources

where they are stored. It is based on a 2-level data structure backed by balanced binary trees. This management data can be searched to locate Resources, e.g. all Resources storing a given set of input data files, or the idle Resource maximizing the Storage Affinity with the input data files of a given Task (ties are broken by selecting the Resource with the most empty data cache in order to balance storage among Resources).

2.7 Scalability of the Data Transfer Architecture

As explained in the previous sections, the BitTorrent-based data transfer overlay is distinct from the Grid overlay. Endpoints from both are connected (see Section 2.3). A P2P Grid is intrinsically scalable, by design: How scalable does it remain when transfers of large input data files occur?

The proposed data transfer architecture is extremely scalable for BitTorrent transfers. The load of BitTorrent data transfers is almost entirely on Resources, not at all on supplier Peers, and only minimally on consumer Peers (see Figure 1 again for an illustration). The proposed use of BitTorrent can be classified as following the BitTorrent hybrid model [5], where the original data source, i.e. the consumer Peer serves data along with other BitTorrent Nodes, i.e. Resources. Consequently, a consumer Peer sharing data files indeed experiences some load, but considerably less than with FTP transfers.

To completely remove the load from consumer Peers, it could be imagined that each consumer Peer first updates the working set of a handful of its Resources with replicas of the input data files that are going to be needed by one of its Tasks soon to be scheduled. The consumer Peer would not schedule any Task to these Resources so that they are used exclusively to share input data files on its behalf. Immediately after these Resources have downloaded the input data files to share, the consumer Peer could entirely stop to share these files, effectively removing from itself any load due to their sharing.

The proposed architecture of data transfers is not immediately scalable for FTP transfers. The load of FTP data transfers is totally on consumer Peers, not at all on supplier Peers and only minimally on Resources (see Figure 1 again). More accurately, the load due to FTP transfers is completely on the FTP servers co-located with consumer Peers.

Better scalability of FTP transfers could be achieved with Content Distribution Networks [27, 31] (CDN). This would however come at the expense of maintain-

ing multiple FTP servers. Interestingly, BitTorrent could be used to synchronize multiple CDN servers efficiently.

3 Task Scheduling

When there are queued Tasks on a Grid Peer, and either some of its Resources or some supplier Peers are available, this Grid Peer schedules as many Tasks as it can. Task scheduling involves selecting (1) which Task to schedule (2) on which Resource (3) of which Grid Peer (4) with which data transfer protocol.

3.1 Task Selection

The order in which Tasks are scheduled, i.e. selected execution, is now discussed. Let $\theta = \theta_0,\ldots,\theta_{n-1}$ be a Bag of n Tasks. Let Δ_i be the set of input data files of Task θ_i, Δ_i^j its j^{th} input data file and $|\Delta_i^j|$ the size (in bytes) of the latter.

Two Tasks θ_i, θ_k are said to be related when they have at least one input data file in common, i.e. $\exists j,l : \Delta_i^j = \Delta_k^l$. A set of Tasks θ is said to be connected if every θ_i is related to at least one other Task, i.e. $\forall i \exists k : \theta_i, \theta_k$ are related. Any BoT can be partitioned into disjoint connected sets of Tasks by repeatedly applying a transitive closure algorithm. A sequence $\sigma(\theta)$ of a connected set θ of Tasks is an ordering of θ. A subsequence $\bar{\sigma_s}(\theta)$ is a section of this ordering, and its length is noted $|\bar{\sigma_s}(\theta)|$. The distance between the sets of input data files of the two Tasks θ_i, θ_k is the sum of the sizes of the input data files of θ_k that are not identical with θ_i: $d(\Delta_i, \Delta_k) = \sum_l |\Delta_k^l|, \forall l : \Delta_k^l \notin \Delta_i$.

To schedule at the same time Tasks sharing some input data files, it could be efficient to minimize the sum of distances between subsequent Tasks within sequences. This would require to take into account the variability of the number of available Resources, and to solve an asymmetric Travelling Salesman Problem [24] for each sequence, which is not desirable in practice. Instead, Tasks sharing input data files can be scheduled at the same time.

The following definitions are introduced to facilitate the discussion. Two Tasks are said to be data-equal when they have all their input data files in common, i.e. $d(\Delta_i, \Delta_k) = 0$. A data-equal subsequence of θ is an extensive subsequence of data-equal Tasks, meaning that the Tasks immediately before and after the subsequence are not data-equal to those belonging to it. The Temporal Tasks Grouping (TTG) algorithm then consists in sorting by decreasing length $|\bar{\sigma_s}(\theta)|$ the data-

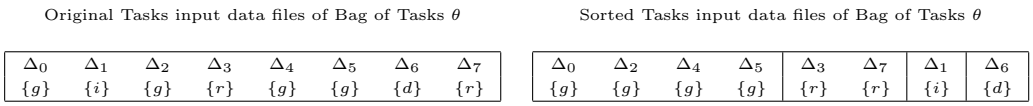

Figure 2: Tasks of Bag of Tasks θ (here with 1 input data file per Task) are grouped into data-equal subsequences, which are sorted by decreasing length $|\bar{\sigma}_s(\theta)|$.

equal subsequences of a sequence of Tasks (see Figure 2). Multiple data-equal subsequences of similar length may then be sorted using a nearest neighbor algorithm [24] (the metric being the distance $d(\Delta_i, \Delta_k)$ between two Tasks of neighbor subsequences).

Importantly, this ordering can be performed similarly to batch-mode [26, 12], but statically at the submission time of the BoT. Indeed, it depends only on the Bag of Tasks itself and does not need any information from other Grid Peers. Actual Task selection is therefore simple and consists in following the precomputed ordering.

The main benefit of Temporal Tasks Grouping is to ensure that the scheduling of data-equal Tasks is temporally grouped so as to maximize efficiency of BitTorrent transfers. Another benefit in the context of a P2P Grid is to schedule first the largest groups of data-equal Tasks. This ensures that large data-equal subsequences, which benefit the most from the use of BitTorrent, are scheduled presumably at a time when there is a large number of available supplied Resources. Upon BoT submission to a Grid Peer, negotiation is immediately performed (if necessary) in order to gather as many Resources to consume as soon as possible. It might be interesting to explore the impact of considering the size of the next group of data-equal Tasks to schedule when negotiating access to Resources of other Grid Peers, in order to negotiate enough Resources to maximize the utility of Temporal Tasks Grouping. It should be noted that, within a given Bag of Tasks scheduled with Temporal Tasks Grouping, the size of the groups of data-equal Tasks is necessarily decreasing.

3.2 Resource Selection

Data placement, or Spatial Tasks Grouping, is explicitly taken into account when selecting a Resource to schedule Local and external Tasks. Let Δ_{R_x} be the contents of the data cache of Resource R_x. At a given time, it contains input data files accumulated from previous Task executions.

When a Grid Peer is scheduling a local or external Task θ_i, a Resource is located by minimizing the distance $d(\Delta_i, \Delta_{R_x})$ between the input data set Δ_i of the Task to schedule and the data cache of each Resource. This distance, computed dynamically,

represents the transfer cost of scheduling θ_i on R_x. It requires data tracking support: Each Grid Peer knows the contents of the data cache of each of its Resources at any time, which is made possible by the way data caches operations have been designed, e.g. the contents of a data cache of a Resource is synchronized on the working set communicated by the Grid Peer that owns the Resource.

The minimum distance will usually be small as there will probably be one Resource with a data cache already storing most of input data files of Δ_i due to the execution of previous Tasks. This minimization is equivalent to maximizing the Storage Affinity [17] (without Task replication) of the given Task with the contents of the data caches of the Grid Peer's Resources.

3.3 Supplier Peer Selection

Data placement, or Spatial Tasks Grouping, is also explicitly taken into account when selecting a supplier Peer. For each BoT that has been submitted by one of its User Agents, a Grid Peer maintains a list of supplier Peers that have supplied Resources for at least one Task of the BoT. For each BoT, a mapping of supplier Peers to the data required by the external Tasks that were submitted to them is also maintained.

Scheduling a local Task to a supplier Peer involves selecting an appropriate supplier. Supplier Peers that already processed other Tasks of the same BoT are ranked first. The ranking is performed using the same distance that was defined in Section 3.2, except that it is applied to the estimated contents of the set of data caches of the target Grid Peer's Resources instead of the data cache of one Resource. Instead, given the context of P2P Grids, the state of the data caches of any other Grid Peers is not available to a given Peer. The mapping between supplier Peers and uploaded data is therefore used as an approximation of this state.

Of course, there are two risks: (1) The expected input data files might have been ejected from the data caches of the Resources of any given supplier Peer, or (2) the Resources of a given supplier Peer storing the data might already be busy when the Task has to be actually scheduled. These risks are unavoidable given the lack of trust between Grid Peers and given the requirement for scalability that excludes a Grid-level monitoring infrastructure.

It is important to highlight the fact that both data-aware Peer selection by consumer Peers and data-aware Resource selection by supplier Peers are needed to provide end-to-end Spatial Tasks Grouping capabilities.

3.4 Data Transfer Protocol Selection

There are several circumstances under which transfer times are longer with BitTorrent than with FTP. The larger overhead comes from choking and the large number of TCP connections to handle. There also is an initial latency caused by the fact that a BitTorrent Node starting to share a file has to wait to obtain permission (i.e. by random chance via so-called optimistic unchoking) to download a few initial pieces before it can start to exchange pieces with other BitTorrent Nodes. Therefore, it would not be efficient to use BitTorrent to share input data files that are small in size or that belong to Tasks of data-equal subsequences that are short in length.

To decide whether to download a given input data file with BitTorrent or FTP, a simplified data transfer protocol selection algorithm is derived [9, 18] from an existing analytical time model [33, 34] of multiple simultaneous transfers of a given file with BitTorrent. The original model gives an estimate of download times. It is a function of the number of BitTorrent Nodes interested in the file to download, file size, network bandwidth and protocol latency, and a factor that is logarithmic in the number of downloaders (i.e. BitTorrent Nodes). Our model is a function of the number of BitTorrent Nodes interested in the file to download and file size.

4 Experimental Results

4.1 Details of Implementation

In a P2P Grid, each Grid Peer hosts its own data server, as opposed to a Desktop Grid, which depends on a centralized data server. Therefore, each Grid Peer runs a BitTorrent client, a BitTorrent tracker, as well as an FTP server, and offers data caching support for the input data files of the Tasks of its User Agents. Likewise, each Resource runs a BitTorrent client and an FTP client, and offers data caching support, enabling Resources to act as transient data servers to other Resources by contributing to BitTorrent data transfers.

The described algorithms and data transfer architecture have been implemented in the Java language (J2SE 5.0). The implementation [11] relies on widely available, off-the-shelf, Free and Open Source data transfer software, deeply integrated as Java libraries: Azureus BitTorrent client and tracker (version 2.5.0.4), Apache FTP server (version 20061027, slightly patched to enhance security by denying several FTP commands), and edtFTPj FTP client (version 1.5.3). It is thus easily deployable as-is on a great number of platforms.

4.2 Useful Metrics

A measure of the diversity (and, conversely, of the redundancy) of input data files between Tasks of a BoT is the Data Diversity Ratio (DDR). It corresponds exactly to the Shared Data Ratio used in recent works [9, 34], but has been renamed to reflect more accurately its semantic. The DDR is defined as the size (in bytes) of all the distinct input data files among all Tasks of the BoT, divided by the total size (in bytes) of all input data files of the Tasks. If there is no sharing of data, DDR = 1. The DDR decreases towards 0 as the level of sharing increases. It has a lower bound that is the size of the smallest file of the BoT divided by the number of bytes of all files in the BoT: $0 < \frac{number\ of\ bytes\ of\ smallest\ file}{number\ of\ bytes\ of\ all\ files} \leq \text{DDR} \leq 1$. For example, the DDR of a Bag of three Tasks depending on the same input data file is 1/3, which also corresponds to the lower bound.

Inter-Task data sharing is a criterion related to data sharing within a BoT. It was introduced by Santos et al. as *"inter-task data reutilization"* [30]. It states that there is data sharing within a BoT whenever at least two Tasks share at least one input data file. It is equivalent to the condition $DDR < 1$.

Inter-BoT data sharing is a criterion related to data sharing between two BoTs. It was introduced by Santos et al. as *"inter-job data reutilization"* [30]. It states that there is data sharing between two subsequently submitted BoTs whenever at least one input or output data file of the firstly submitted BoT is included as one input data file of the secondly submitted BoT. In this paper, Inter-BoT data sharing is considered to be restricted to input data files only because there is currently no support for the caching of output data files in the Lightweight Bartering Grid.

The BoT response time is a widespread metric measuring the time a BoT spends in a distributed computing system. It is defined as the elapsed time (in seconds) between the submission time of a BoT by a User Agent to a Grid Peer and the time when results of all Tasks have been uploaded back to the User Agent. Human users of the Grid typically expect low BoT response times, which constitute an indicator of fast system turnaround.

The cache hit ratio of a Resource is the number of cache hits, divided by the number of cache queries, of the Resource's data cache. It is thus a real number between 0 and 1 included. The mean cache hit ratio of a Grid Peer is the mean of the cache hit ratios of Resources of this Grid Peer. It is thus also a real number between 0 and 1 included. Human administrators of Grid Peers typically expect high mean cache hit ratios, which constitute an indicator of efficient system utilization.

4.3 Deployment

The experiments have been conducted on a cluster of 30 x86 PC (Intel P-IV 3GHz with 1GB RAM), all equipped with standard hard drives, and connected with switched 100 Mbps Ethernet.

The P2P Grids considered in the experiments presented in the next sections may consist of up to: 4 Peers (1 Peer acting in a consumer role, 3 Peers acting in a supplier role), 24 Resources (8 assigned to each supplier Peer), 1 User Agent (submitting Bags of Tasks to the consumer Peer) and 1 search engine. 24 PC (out of 30 available) are used as Resources. 6 PC are used as other Grid nodes: 4 Peers, 1 User Agent and 1 search engine.

In order to avoid interferences arising from queueing issues in the presented experiments, the deployed consumer Peer has no Resource. For the same reason, only the single consumer Peer submits Tasks to the supplier Peers, which have no User Agent and thus are not submitted any local Task. Having only one consumer Peer prevents response times to be influenced by queueing issues.

The activity of each Task consists in simply pausing for 2 seconds and then hashing its associated input data file. These simple Tasks allow to control the impact of Task runtimes on the overall response time of Bags of Tasks.

One large input data file is associated to each Task of each submitted BoT. In case of data redundancy between Tasks, i.e. DDR < 1, Tasks within a submitted BoT are ordered so as to maximize the distance between data-equal Tasks. Identical data files are spread as much as possible within the BoT, which degrades BitTorrent performance the most if a FIFO Task selection algorithm is used instead of TTG. There may exist several such permutations for a given BoT. Multiple BoTs are always submitted sequentially.

Several parameters are taken into account: BoT size, BoT count, input data file size (bytes count), DDR, data transfer protocol, Resource selection algorithm, Task selection algorithm, caching support (file capacity). BoT response times and, when appropriate, mean cache hit ratios (see Section 4.2), are evaluated for each.

4.4 Varying Data Protocol, Task Selection, Caching

Multiple combinations of data transfer protocols, Task selection algorithm and caching support are evaluated. One BoT of 100 Tasks with a medium DDR of 0.25 is submitted, i.e. 25 groups of 4 Tasks depending on an identical input data file, with Tasks depending on the same input data file spread as much as possible in

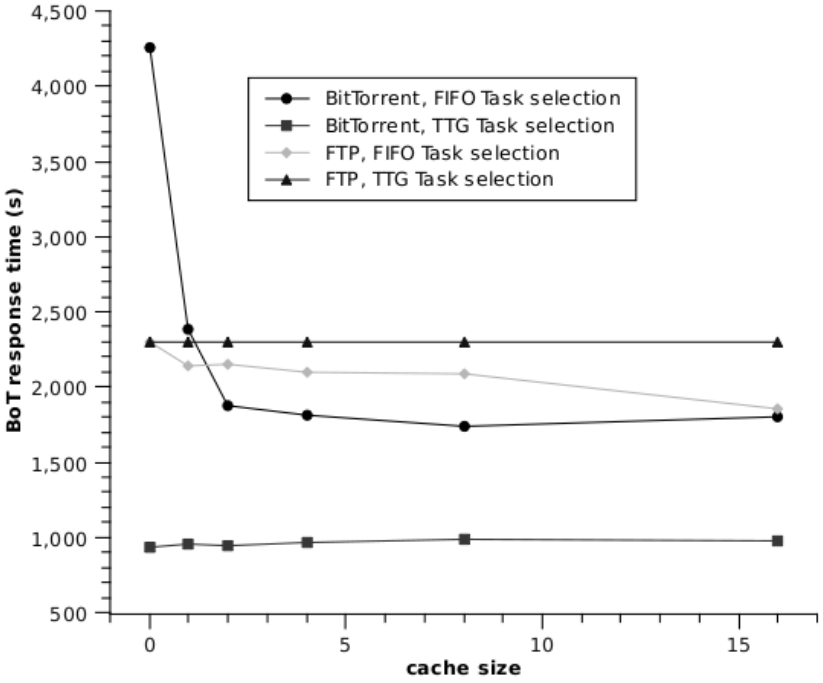

Figure 3: Varying data protocol, Task selection, caching.

the submission order. Results presented in Figure 3 and Table 1 show that in this setup BoT response times are much lower with both BitTorrent data transfers and Temporal Tasks Grouping activated.

In this setup, when Temporal Tasks Grouping is activated, the caching mechanism has no effect, i.e. cache hit ratio equals 0.0, because there is no possibility of cache hits. Indeed, the sets of four Tasks are scheduled concurrently to four separate Resources.

When Temporal Tasks Grouping is not activated, increasing caching support leads to increasing cache hit ratios and decreasing BoT response times, as might be expected. Performance is severely degraded if BitTorrent data transfers are not combined with both caching support and TTG activation.

4.5 Varying Data Protocol, Data Configuration

4.5.1 Comparison of Data Protocols

The impact of file size and DDR on BitTorrent and FTP data transfers is now evaluated. One BoT of 24 Tasks, i.e. exactly one Task per Resource, is submitted so that caching and Resource selection play no role. Resource selection is done with

Fixed parameters	Varying parameters
• 1 BoT of 100 Tasks • file size = 256 MB • DDR = 0.25 • Resource selection = data-aware	• data transfer = { BitTorrent | FTP } • Task selection = { TTG | FIFO } • cache size = { 0 | 1 | 2 | 4 | 8 | 16 }

data transfer = BT, Task selection = FIFO

cache	0	1	2	4	8	16
cache hit ratio	0.0	0.02	0.07	0.09	0.10	0.11
BoT response time (s)	4260	2388	1874	1814	1739	1806

data transfer = BT, Task selection = TTG

cache	0	1	2	4	8	16
cache hit ratio	0.0	0.0	0.0	0.0	0.0	0.0
BoT response time (s)	935	950	948	961	984	972

data transfer = FTP, Task selection = FIFO

cache	0	1	2	4	8	16
cache hit ratio	0.0	0.05	0.06	0.07	0.08	0.18
BoT response time (s)	2297	2137	2146	2093	2092	1850

data transfer = FTP, Task selection = TTG

cache	0	1	2	4	8	16
cache hit ratio	0.0	0.0	0.0	0.0	0.0	0.0
BoT response time (s)	2297	2298	2296	2298	2296	2298

Table 1: Varying data protocol, Task selection, caching.

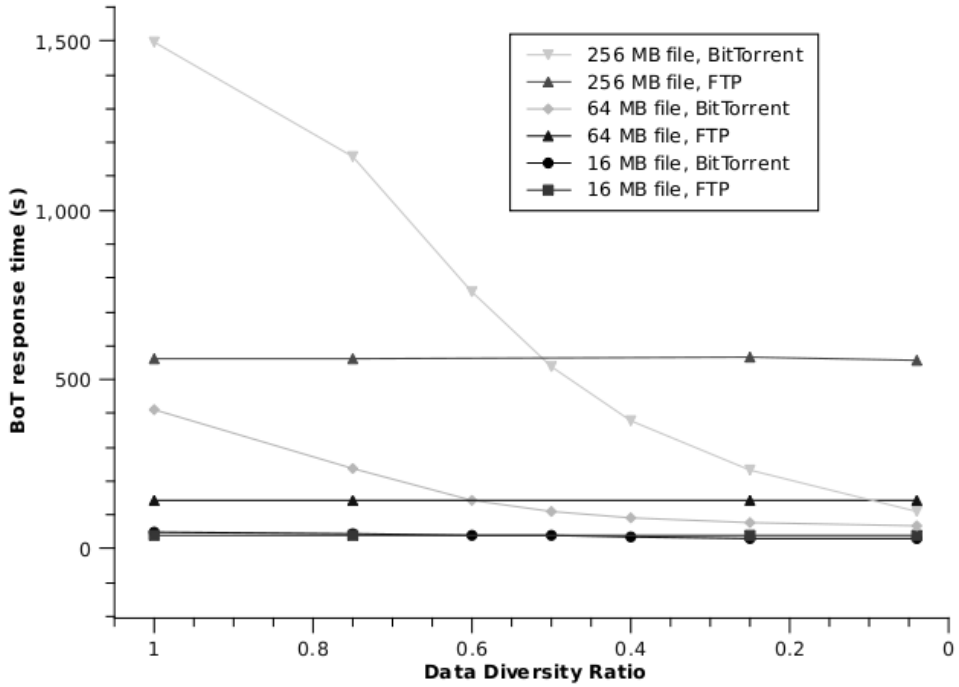

Figure 4: Varying data protocol, data configuration - comparison of data protocols.

Temporal Tasks Grouping.

Results presented in Figure 4 and Table 2 show that BitTorrent data transfers lead to excellent performance in most configurations. They also show that there is a threshold in the range of DDR values beyond which BitTorrent data transfers become more interesting than FTP data transfers (independently of any extra gain from an increased cache size). In the evaluated setup, this threshold is expected to be 0.5 and is indeed close to 0.5. A DDR of 0.5 means that there are 12 independent input data files, with exactly 2 Tasks depending on each of them. A high DDR leads to mediocre performance with BitTorrent data transfers, as there is no opportunity to exploit orthogonal bandwidth. As expected, the results also show that the DDR has no impact on FTP data transfers.

4.5.2 BitTorrent Efficiency with a Small Number of Downloaders

A well-known downside of BitTorrent is its overhead, which is higher than FTP's overhead, as has also been shown in the previous experiment. The performance of BitTorrent with a small number of downloaders and varying levels of caching support is now evaluated. Only one supplier (with 8 Resources) is used in this experiment. A BoT of 100 Tasks depending on an identical input data file is submitted, which is

Fixed parameters	Varying parameters
• 1 BoT of 24 Tasks • Resource selection = data-aware • Task selection = TTG • cache size = 1	• file size = {16 MB | 64 MB | 256 MB} • DDR = { 0.04 | 0.25 | 0.40 | 0.50 | 0.60 | 0.75 | 1.0 } • data transfer = { BitTorrent | FTP }

file size = 16 MB, data transfer = BT

DDR	1.0	0.75	0.60	0.50	0.40	0.25	0.04
BoT response time (s)	49	43	40	38	35	31	30

file size = 16 MB, data transfer = FTP

DDR	1.0	0.75	0.25	0.04
BoT response time (s)	38	38	38	38

file size = 64 MB, data transfer = BT

DDR	1.0	0.75	0.60	0.50	0.40	0.25	0.04
BoT response time (s)	413	235	144	108	93	75	66

file size = 64 MB, data transfer = FTP

DDR	1.0	0.75	0.25	0.04
BoT response time (s)	145	142	141	141

file size = 256 MB, data transfer = BT

DDR	1.0	0.75	0.60	0.50	0.40	0.25	0.04
BoT response time (s)	1497	1160	760	539	377	232	109

file size = 256 MB, data transfer = FTP

DDR	1.0	0.75	0.25	0.04
BoT response time (s)	562	562	565	558

Table 2: Varying data protocol, data configuration - comparison of data protocols.

Fixed parameters (1 supplier only)	Varying parameters
• 1 BoT of 100 Tasks • file size = 256 MB • DDR = 0.01 • Resource selection = data-aware • Task selection = TTG	• data transfer = { BitTorrent | FTP } • cache size = { 0 | 1 }

data transfer	BT	BT	FTP	FTP
cache	0	1	0	1
cache hit ratio	0.0	0.92	0.0	0.92
BoT response time (s)	2331	201	2300	303

Table 3: Varying data protocol, data configuration - BitTorrent efficiency with a small number of downloaders.

typical of parameter sweeps applications. Temporal Tasks Grouping and data-aware Resource selection are both activated.

Results in Table 3 first show that having caching support, even very limited, brings considerable performance gains. With limited caching support, the data-aware Resource selection and Temporal Tasks Grouping algorithms bring their benefits: BitTorrent outperforms FTP, even with only 8 downloaders.

4.6 Inter-Task Data Sharing, Varying Task Selection

The impact of Temporal Tasks Grouping is now evaluated. A BoT of 48 Tasks, which is twice the number of available Resources, is submitted. Different levels of data sharing are tested, with and without activation of TTG. Two observations can be made about results presented in Figure 5 and Table 4.

First, all else being equal, BoT response times obtained with TTG-based Resource selection are much better, i.e. shorter, except for a very low DDR. In this latter case, caching plays a greater role, leading to BoT response times roughly equivalent between TTG-based and FIFO-based Resource selection. The difference between the two policies is that network bandwidth is traded for storage space.

Second, caching has a huge impact, even at very low levels of cache hit ratios. Deploying Resources with limited caching support has a strong practical interest due to the combination of Data-Intensive BoT and the use of edge Resources.

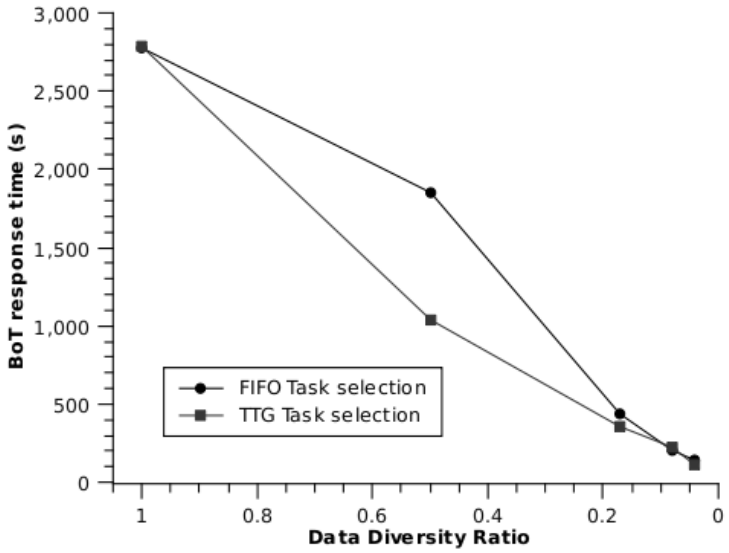

Figure 5: Inter-Task data sharing, varying Task selection.

Fixed parameters	Varying parameters
• 1 BoT of 48 Tasks • file size = 256 MB • data transfer = BitTorrent • Resource selection = data-aware • cache size = 6	• DDR = {0.04 | 0.08 | 0.17 | 0.50 | 1.0} • Task selection = { TTG | FIFO }

Task selection = FIFO

DDR	1.0	0.50	0.17	0.08	0.04
cache hit ratio	0.0	0.02	0.07	0.33	0.61
BoT response time (s)	2775	1854	442	201	142

Task selection = TTG

DDR	1.0	0.50	0.17	0.08	0.04
cache hit ratio	0.0	0.0	0.0	0.04	0.38
BoT response time (s)	2783	1039	361	223	113

Table 4: Inter-Task data sharing, varying Task selection.

4.7 Inter-BoT Data Sharing, Varying Resource Selection

A common and widespread scenario is that of a scientist who sequentially submits multiple times the same BoT with high DDR - i.e. there is little or no inter-Task data sharing - and the same algorithm run by all Tasks. Only the input parameters or the algorithm are modified between consecutive executions of the BoT after an out-of-Grid analysis of computed results, i.e. an experiments session. The input data files of a given Task thus remain the same from one execution of the BoT to another.

An experiment is run for this relevant scenario. Eight Bags of 24 Tasks are submitted. Data-aware Resource selection is varying between fully activated, activated for Peers only, activated for Resources only, and completely deactivated. Results in Figure 6 and Table 5 show that the cache hit ratio increases with the level of data-awareness in the Resource selection. It is optimal, i.e. 7/8 in this experiment, when both Peer-level and Resource-level data-aware Resource selection algorithms are activated. BoT response times also decrease sharply when the cache hit ratio increases.

5 Discussion

5.1 BitTorrent vs. GridFTP

5.1.1 Why BitTorrent Is Relevant

The relevance of using BitTorrent - rather than other P2P file sharing protocols - in P2P Grids is now discussed.

- First, BitTorrent is both highly scalable [32] and highly adaptive to unreliable network conditions, making it suitable for P2P Grids.
- Second, the built-in BitTorrent incentive mechanism discourages malicious Grid nodes to run misbehaved BitTorrent Nodes.
- Third, piece-level data exchanges enable Temporal Tasks Grouping, whereas file-level data exchanges would not.
- Fourth, the default behavior upon completion of a download is to continue sharing the downloaded file. This behavior actually encourages Grid Peers to schedule Tasks using Temporal Tasks Grouping as this has the effect of maintaining a

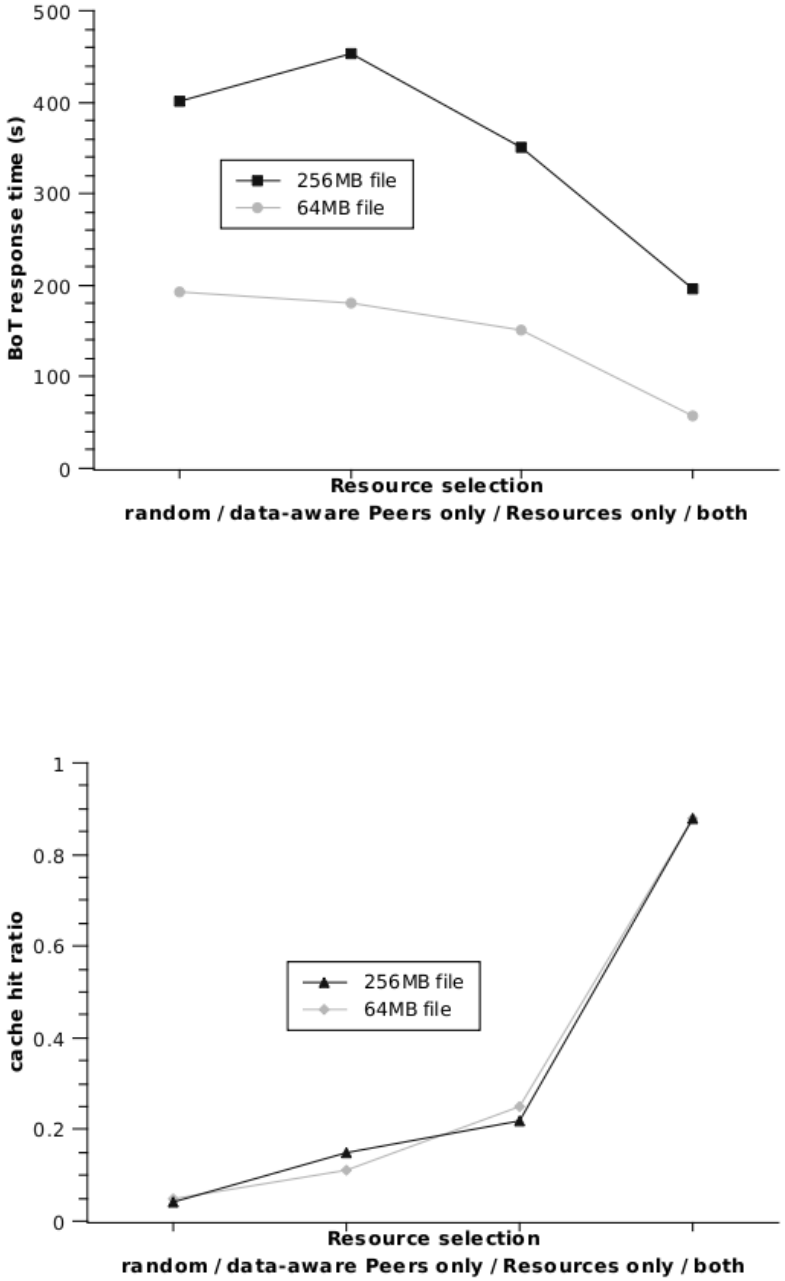

Figure 6: Inter-BoT data sharing, varying Resource selection.

Fixed parameters	Varying parameters
• 8 BoT of 24 Tasks each • DDR = 1.0 • data transfer = BitTorrent • Task selection = TTG • cache size = 1	• file size = { 64 MB | 256 MB } • Resource selection = { random | data-aware for Peers only | data-aware for Resources only | data-aware }

file size = 64 MB

Resource selection	random	data-aware Peers only	data-aware Res. only	data-aware
cache hit ratio	0.05	0.11	0.25	0.88
BoT response time (s)	192	181	151	56.5

file size = 256 MB

Resource selection	random	data-aware Peers only	data-aware Res. only	data-aware
cache hit ratio	0.04	0.15	0.22	0.88
BoT response time (s)	401	453	350	196

Table 5: Inter-BoT data sharing, varying Resource selection.

large number of data sources even if a flash crowd is not totally synchronized. It is important to remark that, when supplementary storage space is needed, a Grid node will eventually stop sharing a completely downloaded input data file not needed by the currently running Task, thus effectively precluding downloaded files from saturating the storage capacities of a P2P Grid.

- Fifth, the BitTorrent Node discovery service, i.e. the BitTorrent tracker, operates at an appropriate level of granularity, as each Grid Peer can operate its own BitTorrent tracker to manage a discovery service for its own files. Indeed, as explained in Section 6.2.3 there is no need for a global Grid-level tracker that would manage a BitTorrent discovery service for all files of a P2P Grid.

Introducing QoS in BitTorrent [5, 23] has been proposed in recent research work in Content Networks, which are very relevant to the context of P2P Grids. This may lead to variants of BitTorrent enhanced for usage in P2P Grids.

5.1.2 Why GridFTP Is Not Relevant

GridFTP [3] is an extended FTP protocol specifically designed for high performance, highly managed Grids.

Historically, GridFTP has targeted controlled, high performance environments in general, and cluster-to-cluster file transfers in particular. A key strength of GridFTP is striping support. Striping is the ability to perform striped (parallel) data transfers of a file through several network interfaces. *"Data striping refers to the segmentation of logically sequential data, such as a single file, so that segments can be assigned to multiple physical devices [...] and thus written concurrently"* [35].

- But striping support, which is a key strength of GridFTP, is of little value in a P2P Grid context. Clearly, it cannot be used in a P2P environment because P2P networks are constituted of edge computers, rarely with more than one active network interface or a lot of available network bandwidth.

- More importantly, GridFTP does not exploit orthogonal bandwidth, i.e. the network links between downloaders, whereas it is the key strength of BitTorrent.

Despite all its strengths, GridFTP is not an appropriate technology for multiple simultaneous transfers of the same data file in a P2P Grid. It cannot be used as a baseline protocol of a data transfer architecture in a P2P Grid, but could be used as a secondary protocol to optimize some BitTorrent operations. As Allcock et al. [3]

have pointed out, GridFTP *"could be used to good effect as a data transfer tool in"* BitTorrent to augment the reliability and performance of TCP/IP connections between BitTorrent Nodes.

5.2 Impact of the Involved Technologies

5.2.1 Impact of BitTorrent

If no BitTorrent support is available, TTG is not useful and the only gains of performance would come from caching and data reuse. BitTorrent must be activated for TTG to have a favorable impact on transfer times of identical input data files. Reciprocally, excellent performance of BitTorrent data transfers is achieved only if the scheduling of Tasks with identical input data files is temporally grouped.

FTP should be activated instead of BitTorrent to transfer small files or files of a BoT where the number of identical files is limited.

5.2.2 Impact of Temporal Tasks Grouping

Task selection with the Temporal Tasks Grouping algorithm temporally groups the scheduling of Tasks with identical input data files. This maximizes the number of simultaneous BitTorrent Nodes downloading identical files, thereby dramatically decreasing the mean file transfer times.

The activation of TTG has no impact on FTP transfers: The order of some direct data transfers from the consumer Peer is merely different from the initial order between Tasks in the BoT.

5.2.3 Impact of Caching

Caching of data on Resources naturally brings important benefits as it can prevent unnecessary data transfers. A larger cache capacity brings even greater benefits. On the downside, caching has a cost in terms of storage space. However, it is important to highlight that even limited caching support already brings considerable benefits.

Even in the presence of strong caching support, TTG is very important in case of inter-Task data sharing within a BoT. TTG is indeed the only means for efficient data transfers of files that are introduced in the Grid for the first time.

Conversely, when TTG is not activated or has a limited effect - i.e. for BoTs with a high DDR or if BitTorrent is not activated - caching is very important.

It should also be noted that the proposed management of data caches is fully dynamic, and thus more adapted to P2P Grids than existing data cache management policies that are statically precomputed [28].

5.2.4 Impact of the Cache Replacement Policy

Using a pure LRU cache replacement policy is sub-optimal. Better performance would probably be achieved by taking into account the size of the cached files.

For example, let us consider a P2P Grid of 24 Resources, each having a cache size of 2 files. Those Resources can use two different cache replacement policies: pure LRU (the least recently used file in the cache is ejected first) or weight-based (the smallest file in the cache is ejected first). Task selection is based on Temporal Tasks Grouping and Resource selection is data-aware.

3 BoTs of N Tasks (θ_1, θ_2, θ_3) are submitted sequentially. All Tasks of a given BoT depend on an identical file (Δ_1^0, Δ_2^0, Δ_3^0 respectively): The DDR of each BoT is thus $\frac{1}{24}$. Let us assume as an example that Δ_1^0 weighs 200 MB; Δ_2^0 and Δ_3^0 both weigh 1 MB.

With Temporal Tasks Grouping, all Resources hold both Δ_1^0 and Δ_2^0 in their cache after the completion of θ_2. When θ_3 is submitted, depending on the cache replacement policy (LRU or weight-based), either Δ_1^0 (the oldest) or Δ_2^0 (the smallest) is ejected from the caches and replaced by Δ_3^0. At this point, cache hit ratios are all equal to zero (for both policies) since a different file was required for every Task.

Let us hypothesize that θ_1 is then resubmitted. Depending on the cache replacement policy (LRU or weight-based), Δ_1^0 has to be downloaded again (which is expensive since it is a 200 MB file) or is loaded from the caches (in this case, yielding a cache hit ratio of 0.25). With LRU, the BoT response time will be larger and the perceived performance will be lower.

It can be argued that a purely weight-based replacement policy is not optimal either. A hybrid cache replacement policy, maybe also involving additional factors, could lead to even better performance.

5.2.5 Impact of Data-Aware Resource/Peer Selection

The impact of data-aware supplier Peer selection is that the supplier Peer possibly storing the maximum number of input data files required by a scheduled Task is selected. Additionally, the impact of data-aware Resource selection is that the Resource of the selected supplier that stores the maximum number of required files is

guaranteed to be selected.

Deactivating data-aware Resource selection or supplier Peer selection severely degrades BoT response times and cache hit ratio when scheduling BoTs with a low DDR, i.e. highly identical input data files. Reciprocally, data-aware Resource selection and Peer selection are useless without caching support. For a BoT exhibiting a high DDR, i.e. there is no inter-Task data sharing, data-aware Resource or supplier Peer selection has a strong practical interest in the case of inter-BoT data sharing.

5.2.6 Performance of BitTorrent Data Transfers

Temporal Tasks Grouping enables the excellent performance of BitTorrent data transfers in situations of inter-Task data sharing, i.e. when several Tasks of a given BoT share input data files.

Caching support also increases performance of BitTorrent transfers in situations of inter-BoT data sharing, i.e. when Tasks of multiple, successive BoTs share input data files. Therefore, caching support can be viewed as an indirect way to increase the number of BitTorrent Nodes sharing input data files. This is very useful as it supports opportunistic scheduling of Tasks to extra supplementary Resources as they become available. This could be summarized by stating that the effect of TTG on performance of BitTorrent transfers is immediate, while the effect of caching support is temporally delayed.

5.3 Recommended Deployment Options

Maximization of data reuse is enabled by combining caching support and data-aware Resource and supplier Peer selection. Caching support should be configured to use as much storage space as possible on each Resource.

Temporal Tasks Grouping and BitTorrent support enable to maximize parallelism of Tasks execution. BitTorrent support should be made systematically available, and Temporal Tasks Grouping should be systematically activated. It should be emphasized that the deployment of BitTorrent software does not mean that all transfers will be performed with BitTorrent, but only that it is available. Peers can be configured to use BitTorrent either systematically, not at all or following an adaptive data transfer protocol selection algorithm.

If Resources of a given Grid Peer exhibit large variation in reliability or performance, Task replication [30] or a performance- or reliability-aware [10] Resource selection algorithm should be used.

6 Related Work

6.1 Scheduling Bags of Tasks

Task scheduling essentially consists in matching queued Tasks and available Resources. Classic Task scheduling strategies [6, 13, 16, 17, 19, 29, 30] data replication and Spatial Tasks Grouping, are first presented. Temporal Tasks Grouping, a novel strategy that we have recently proposed [9] to enable high levels of Task execution parallelism, is also presented.

6.1.1 Data Replication

A possibility to avoid that massive data transfers degrade performance is to perform them when it will not increase Bag of Tasks response times. Proactively replicating data on multiple Resources may increase the efficiency of subsequent data-aware Task scheduling, but with a possibly large storage cost due to data caching.

In relatively stable environments, more typical of Desktop Grids rather than of P2P Grids, it is possible to complete all data replication before scheduling a BoT. A recent example of state-of-the-art synchronous data replication in a Desktop Grid, in the context of life sciences applications [19], proposes an integer programming scheduling algorithm. It is limited to a steady state context. It is therefore not applicable in the context of P2P Grids because of the unreliability and constant change of P2P environments.

It has been shown [29] that data replication can be performed proactively and asynchronously from Task scheduling, with simple and cost-efficient algorithms, as long as the Task scheduler is aware of data placement. As proactive data replication in a very dynamic environment will seldom achieve perfect data placement, a trade-off [8] must be found between creating new replicas in order to balance the computational load and using existing replicas in order to avoid data transfers.

6.1.2 Spatial Tasks Grouping

The costs incurred by data replication can also be very high in terms of transfer times and of infrastructure overload. The overload of a consumer Peer may lead to performance bottlenecks due to the thrashing caused by the handling of data management threads or by the saturation of the network bandwidth.

Another Task scheduling strategy is therefore to avoid data replication by simply relying on data caching. If each Resource is equipped with a data cache, only the

first Task requiring a particular data file will trigger its transfer. Subsequent Tasks requiring the same data file will not trigger its transfer if it is still present in the data cache. Data caching, however, is of limited use if Task scheduling does not take account of data placement.

When prior availability of data on Resources is taken into account, Task scheduling is said to be data-aware. The goal of data-aware Task scheduling is to maximize data reuse and prevent unnecessary transfers of input data files. In this paper, for clarity, the term Spatial Tasks Grouping designates the strategy which consists in combining data caching and awareness of data placement. Task execution parallelism is naturally reduced when following the Spatial Tasks Grouping strategy.

A Task scheduling algorithm [17] has been proposed to take advantage of data caching. It sequentially schedules to the same Resources the Tasks requiring identical input data files.

Another proposed Task scheduling algorithm [17] schedules to the same Resource all Tasks requiring the same set of large data files, which of course decreases execution parallelism but could prove to be very efficient in P2P Grids connected with slow data links, or when the data files are so large that it would not be practical to transfer them more than once [7].

A proposed Storage-aware Workflow scheduling algorithm [28] is based on a statically precomputed schedule of data reuse and cache cleaning operations.

The Storage Affinity [30] Task scheduling algorithm available in the OurGrid [13] P2P Grid middleware dynamically takes data placement into account and is thus perfectly adapted to P2P Grids. It schedules Tasks first to computational resources where most of the required input data files are already available before considering other resources where the unavailability of some input data files requires data replication. The data transfer architecture presented with Storage Affinity [30] does not rely on multiple Resource-level data caches per Peer, but on a single Peer-level data cache on each Peer. The Peer-level data cache is accessed by Resources through an NFS file system, which is not highly scalable.

As it is operating in a P2P environment, Storage Affinity proposes to deal with the unavailability of accurate data about computational times. Task replication [30] is proposed as a heuristic to find good Task-to-Resource assignments. Redundancy brings excellent tolerance to Resource faults and variability in performance, but at a cost [14]. It can be remarked that Task replication has the side effect that any BoT *de facto* becomes Data-Intensive, in the sense that its input data files are transferred multiple times.

As Spatial Tasks Grouping depends on the presence of data caching support, its cost in terms of storage space on Resources should not be neglected. Indeed, on one hand, files processed by Data-Intensive BoTs are inherently large and, on the other hand, storage capacity on Resources in a P2P Grid are inherently variable and potentially limited.

6.1.3 Temporal Tasks Grouping

Task selection can be done so as to increase the parallelism of Task execution on multiple Resources, A performance bottleneck may arise if flash crowds are not handled efficiently.

As opposed to other related protocols, the BitTorrent P2P file sharing protocol is able to prevent such performance bottlenecks. Most of the few recent works studying the use of BitTorrent-based data transfers [9, 21, 33, 34], exhibit deployment constraints that are not adapted to P2P Grids: Either data scheduling is centralized [33, 34] and requires that all Resources follow a statically defined data transfer schedule (this is called Predictive Communications Ordering), or Task scheduling is very basic and unadapted to P2P Grids [21].

We have recently proposed the Temporal Tasks Grouping [9] Task scheduling algorithm. Any Data-Intensive Bag of Tasks may be partitioned into several groups of Tasks, with each group depending on a different subset of input data files. Temporal Tasks Grouping relies on BitTorrent to perform data transfers. Its goal is to simultaneously schedule a maximum number of Tasks needing the same input data files. In order to exploit the efficiency of BitTorrent, Temporal Tasks Grouping provokes flash crowds by minimizing the time span during which any given file will be required by the Resources of a P2P Grid.

6.1.4 Tasks Execution Parallelism vs. Data Reuse

In existing research, Task execution parallelism and data reuse are usually presented as mutually exclusive.

Great parallelism in Task scheduling, leading to lower overall BoT response times, can be achieved by simultaneously scheduling a large number of Tasks. However, the simultaneous transfer of the input data files of several Tasks scheduled at the same time rapidly leads to performance bottlenecks.

Spatial Tasks Grouping entirely avoids the problem of multiple simultaneous data transfers, leading to a lower impact of data transfers. On the downside, Tasks

- notably those sharing the same input data - are scheduled sequentially rather than simultaneously. This leads to higher BoT response times and it is more difficult to offer nontrivial QoS [20]. Moreover, in practice it is not always possible to massively use data caching as the availability of storage space on P2P Resources is highly variable and may be quite limited.

To achieve a good level of performance in all situations, both Task execution parallelism and data reuse should be activated. Temporal Tasks Grouping [9] and BitTorrent must be combined to prevent performance bottlenecks that arise from multiple simultaneous data transfers.

6.2 Distributed Data Transfers

6.2.1 Optimized Direct Data Transfers

Grids exhibit certain features that can be taken advantage of, such as high speed networks and the presence of multiple network interfaces on many Grid nodes. Some direct data transfer protocols have been modified specifically for use in Grids with the goal to speed up massive data transfers.

GridFTP [3] is *"a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. It is based upon the Internet FTP protocol, and it implements extensions for high-performance operation that were either already specified in the FTP specification but not commonly implemented or that were proposed as extensions by [the Globus alliance]."*

6.2.2 Group-based Data Transfers

Another possibility to efficiently transfer a given data file simultaneously to multiple downloaders is to use multiple sources of data sharing, which can be seen as a variant of data replication. Using a set of data caches scattered over a P2P Grid leverages the bandwidth of several Peers and partially redistributes the transfer load across the P2P Grid. Recent work includes the Super-Peer model [16], and also the File Mover overlay network [6]: *"An overlay network is a computer network which is built on top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. For example, many P2P networks are overlay networks because they run on top of the Internet"* [35]. Both the Super-Peer Model and File Mover require the explicit deployment of Peer-independent data

caches as well as of a routing substrate, the overlay.

Yet another, even more efficient possibility to efficiently transfer a given data file simultaneously to multiple downloaders is that downloaders act cooperatively as a group with a common interest.

The BitTorrent [15, 25] P2P file sharing protocol is a protocol that enforces the cooperative behavior of a group of downloaders with a common interest. Other P2P file sharing protocols are available, but BitTorrent exhibits several features that are useful in a P2P Grid environment. First, BitTorrent P2P networks are unstructured P2P networks that do not need the centrally controlled construction of an overlay on top of Peers and, as such, are easy to deploy. Second, collaboration between downloaders starts very early, as pieces of files, rather than whole files, are exchanged. Third, BitTorrent already has a record of successful integration with Grid middlewares [21, 33, 34].

6.2.3 An Overview of BitTorrent

BitTorrent [15, 25] enables computers running the BitTorrent client software (called BitTorrent Peer) to exchange files with one another in a P2P fashion. As opposed to what happens in other P2P file sharing protocols, a BitTorrent P2P network comes into existence for a single file only, i.e. a given BitTorrent Peer downloading two files in parallel is a member of two BitTorrent P2P networks.

It should be noted that some Grid Peers may never exchange files with BitTorrent, thus they will never become BitTorrent Peers, i.e. they will never become members of any BitTorrent P2P network. Some Resources may exchange files with BitTorrent, making them BitTorrent Peers of BitTorrent P2P networks. Consequently, two types of Grid nodes, i.e. Grid Peers and Grid Resources, may act as BitTorrent Peers. As BitTorrent is used by Grid nodes, two overlays are implicitly coexisting: a Grid overlay and a BitTorrent overlay.

A BitTorrent Node that shares a complete file with other BitTorrent Nodes is called a seeder. It must first split the file to share into pieces and create so-called torrent metadata describing this file as well as the location of the tracker that will be used.

The BitTorrent tracker software is the file-level BitTorrent Node discovery service, which introduces to one another BitTorrent Nodes interested in a given file. A BitTorrent Node sharing files must either launch its own tracker or use a publicly available tracker. There are potentially as many trackers as there are Grid Peers.

Each tracker manages the discovery service of all files shared by the Grid Peer that operates it.

As each Grid Peer runs a BitTorrent tracker, the proposed data transfer architecture is already very scalable in this regard. The location data (i.e. host name, network address) of the BitTorrent tracker used for a given file has to be communicated somehow to BitTorrent Nodes that want to download the given file. Trackerless versions of the BitTorrent protocol are beginning to appear. Given their recency, it is not clear yet whether they are resistant to sabotage by malicious BitTorrent Nodes attempting to corrupt or to slow down the propagation of, information that would be communicated by the BitTorrent tracker.

Each BitTorrent Node that wants to download a given file initially downloads a first piece from any of the BitTorrent Nodes communicated by the tracker. It then begins to exchange pieces with other BitTorrent Nodes. A Peer invites other BitTorrent Nodes to cooperate by allowing them to download its pieces.

As opposed to several other P2P file sharing protocols (Napster, Gnutella, ...), BitTorrent Nodes do not have to wait for a file to be completely downloaded to begin uploading some of its pieces to other BitTorrent Nodes. Indeed, the BitTorrent protocol enables BitTorrent Nodes to simultaneously act as downloaders and uploaders from the very beginning, i.e. as soon as one piece of the file has been downloaded. The BitTorrent protocol also specifies the default behavior that each BitTorrent Node continues to act as uploader after its role as a downloader is finished, i.e. sharing of a given file continues after its download has been completed. As it will be explained, this behavior is interesting in the context of P2P Grids.

With BitTorrent, as opposed to what happens with direct file transfer protocols, network links between BitTorrent Nodes are exploited: As each downloader is also an uploader, the network load is removed from the seeder and distributed to participating BitTorrent Nodes. BitTorrent is able to exploit so-called orthogonal network bandwidth, which is made up of the *"physical network paths not included in a source-rooted application level tree."* [1] The resulting advantage over direct file transfer protocols, e.g. File Transfer Protocol (FTP), is that the total transfer time of a file by multiple downloaders increases slowly with their number [33], i.e. less than linearly. This property makes BitTorrent able to handle efficiently flash crowds of downloaders.

Very recent studies [1, 2, 22, 36] on the use of BitTorrent in Grid environments show that BitTorrent performs nearly as well as explicit overlay-based technolo-

gies in over-provisioned network cores. More importantly, they also indicate that BitTorrent sustains *"equivalent undegraded performance"* as the available network bandwidth decreases. In other words, BitTorrent performs well in managed and over-provisioned networks and, as opposed to explicit overlay-based techniques, maintains good performance in bandwidth-constrained networks that are more typical of P2P Grids environments.

Transferring data with the BitTorrent P2P file sharing protocol in a P2P Grid opens a very interesting perspective: As more Tasks sharing the same input data files are scheduled at the same time, the overall cost of the multiple simultaneous data transfers remains close to the cost of transferring it only once. In turn, this enables the use of Temporal Tasks Grouping, without excluding the use of Spatial Tasks Grouping.

6.2.4 BitTorrent Support in Various Grid Types

The idea of using the BitTorrent file sharing protocol in distributed computing systems has been proposed in several research works. BitTorrent has first been introduced in stable distributed environments (Desktop Grids and clusters) usually as loosely coupled, script-based implementations [21], but also as a well-integrated, but specially modified, BitTorrent library [33, 34]. BitTorrent has also been considered, but not yet implemented, in mainstream Grids and Volunteer Grids. We have recently proposed the use of BitTorrent in P2P Grids [9], relying on an off-the-shelf BitTorrent library deeply integrated with the P2P Grid middleware.

Desktop Grids and Clusters Using BitTorrent in a Desktop Grid has been recently and independently proposed in two studies, one by Wei, Fedak and Cappello [34], and one by us [21].

In the first mentioned study [34], a model of BitTorrent transfer times is described (building upon previous work [33]), along with a BitTorrent data transfer architecture and BitTorrent-aware Task scheduling.

Several BitTorrent-aware versions of classic, knowledge-based scheduling heuristics (BT-MinMin, BT-MaxMin, BT-Sufferage) are proposed. The BT-X knowledge-based scheduling heuristics [34] are designed to operate in a cluster or Desktop Grid environment. They require *"knowledge about communication performance and CPU load performance."* However, this data is not generally easy to obtain, and it is specifically not to be trusted in a P2P environment.

Furthermore, the proposed heuristics use a modified version of BitTorrent. The standard algorithm used to select with which BitTorrent Nodes data will be exchanged - called the choking algorithm because it is said to unchoke BitTorrent Nodes - is based on a loose reciprocity-based policy, i.e. tit-for-tat. It has been replaced by Predictive Communications Ordering (PCO). This is only possible when Grid Peers downloading input data can be centrally managed, which is not the case in P2P Grids. For these two reasons (requirement of hard-to-obtain knowledge and PCO), the BT-X heuristics cannot be applied to the context of P2P Grids.

This very interesting work is, to the best of our knowledge, the first published proposal to combine Task scheduling and BitTorrent data transfers.

In the second mentioned study [21], a Computer Vision Learning problem, structured as a Data-Intensive Bag of Tasks application, is presented. This BoT is shown to be successfully computed with a non-dedicated Desktop Grid running a basic, proprietary middleware. In this context, BitTorrent is used to simultaneously transfer half-gigabyte-sized data to several dozens of Resources.

Finally, loosely coupled BitTorrent support has been implemented[2] in a computer cluster but, again to the best of our knowledge, there is no relevant publication.

Volunteer Grids BitTorrent support for data transfers in Volunteer Grids has been considered [4] but, to the best of our knowledge, there is no relevant publication that proposes a complete mechanism or architecture.

Mainstream Grids Recent research [1, 2] has shown through simulation that BitTorrent is indeed an excellent choice of data transfer technology in networks that are typical of a P2P Grid environment. It has also been shown that in the near future BitTorrent will become an excellent choice in networks that are typical of a mainstream Grids.

The main goal of this recent research was to investigate the possibility of using BitTorrent in highly managed Grids, such as those running gLite in the context of the CERN LHC experiments. A conclusion is that BitTorrent support will be required in future mainstream Grids as the deluge of data to process increases year after year and will eventually overcome the capacity of every existing network. This conclusion is supported by a recent BitTorrent-like protocol, called GridTorrent [22], that has been designed specifically for mainstream Grids.

[2] `http://www.umiacs.umd.edu/~nedwards/research/bittorrent_for_clusters.html`

P2P Grids The environment of P2P Grids is different from the stable environment of Desktop Grids, clusters and mainstream Grids. A data transfer architecture based on BitTorrent is even more relevant in P2P Grids. Our recent work [9] has proposed and shown how to use BitTorrent in P2P Grids.

7 Conclusion

The research presented in this paper can be summarized as combining P2P computing with P2P file sharing technologies. The issue of scheduling Data-Intensive Bags of Tasks in P2P Grids is addressed by combining algorithms enabling both data reuse and high parallelism of Task execution.

Several existing state of the art research works in Grid Task scheduling [11, 13] and P2P Grid Task scheduling (Storage Affinity algorithm [30] and Super-Peer model [16]) have been synthesized.

Caching support is used jointly with data-aware Resource and supplier Peer selection scheduling algorithms in order to exploit the locality principle. This maximizes data reuse and avoids unnecessary transfers of recently downloaded data files. However, the storage costs of caching can be very high, and sets of input data files not previously downloaded in the P2P Grid do not benefit from data reuse.

The main contribution of this paper is our proposal to use the BitTorrent P2P file sharing protocol jointly with a novel Task selection scheduling algorithm (Temporal Tasks Grouping), which generates flash crowds in a controlled way in order to synchronize data transfers on Tasks schedule.

Another major, original contribution is our Java (J2SE 5.0) implementation.[3] To the best of our knowledge, it is the first to propose the joint use of P2P file sharing and P2P computing technologies, relying on a standard, off-the-shelf BitTorrent library. It relies exclusively on Free and Open Source data transfer software (Azureus, Apache FTP server, edtFTPj). The proposed data transfer architecture is highly scalable and adaptive and it does not need Predictive Communications Ordering or an explicit deployment of an overlay network. It is also easily deployable.

The next step in the evolution of the research presented in this paper is to evaluate the suggested deployment options with the Lightweight Bartering Grid middleware running on large-scale P2P Grids. It will also be interesting to evaluate the integration of reliability- and performance-aware Resource and supplier Peer

[3] `http://www.montefiore.ulg.ac.be/~briquet/`

selection algorithms with the current data-aware selection algorithms. This will enable us to tackle issues arising from large variations in reliability and performance of Grid nodes.

Acknowledgement

We want to thank Elisabeth Spilman for the proofreading. We also want to thank the reviewers for their helpful suggestions. Figure 1 includes icons from the Tango library[4] under Creative Commons Attribution Share-Alike license.

References

[1] Samer Al Kiswany and Matei Ripeanu. A Simulation Study of Data Distribution Strategies for Large-scale Scientific Data Collaborations. In *Proc. CCECE*, Vancouver, BC, Canada, April 2007.

[2] Samer Al Kiswany, Matei Ripeanu, Adriana Iamnitchi, and Sudarshan Vazhkudai. Are P2P Data-Dissemination Techniques Viable in Today's Data Intensive Scientific Collaborations ? In *Proc. Euro-Par*, Rennes, France, August 2007.

[3] William Allcock, John Bresnahan, Rajkuma Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus Striped GridFTP Framework and Server. In *Proc. SC*, Seattle, WA, USA, 2005.

[4] David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proc. Grid*, Pittsburgh, PA, USA, November 2004.

[5] Nazareno Andrade, Jaindson Santana, Francisco Brasileiro, and Walfredo Cirne. On the Efficiency and Cost of Introducing QoS in BitTorrent. In *Proc. GP2PC*, Rio de Janeiro, Brazil, May 2007.

[6] Cosimo Anglano and Massimo Canonico. The File Mover: An Efficient Data Transfer System for Grid Applications. In *Proc. CCGrid*, Chicago, IL, USA, 2004.

[7] Michael Beynon, Renato Ferreira, Tahsin Kurc, Alan Sussman, and Joel Saltz. DataCutter: Middleware for Filtering Very Large Scientific Datasets

[4] `http://tango.freedesktop.org/`

on Archival Storage Systems. In *Proc. IEEE Symposium on Mass Storage Systems*, College Park, MD, USA, March 2000.

[8] Michael Beynon, Tahsin Kurc, Alan Sussman, and Joel Saltz. Optimizing Execution of Component-based Applications using Group Instances. In *Proc. CCGrid*, Brisbane, Queensland, Australia, May 15-18 2001.

[9] Cyril Briquet, Xavier Dalem, Sébastien Jodogne, and Pierre-Arnoul de Marneffe. Scheduling Data-Intensive Bags of Tasks in P2P Grids with BitTorrent-enabled Data Distribution. In *Proc. UPGRADE-CN'07, HPDC Workshops*, Monterey Bay, CA, USA, June 2007.

[10] Cyril Briquet and Pierre-Arnoul de Marneffe. Learning Reliability Models of Grid Resource Supplying. In *Proc. Cracow Grid Workshop*, Cracow, Poland, 2005.

[11] Cyril Briquet and Pierre-Arnoul de Marneffe. Description of a Lightweight Bartering Grid Architecture. In *Proc. Cracow Grid Workshop*, Cracow, Poland, 2006.

[12] Massimo Canonico. *Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids*. PhD thesis, University of Torino, Italy, Torino, Italy, 2006.

[13] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro B. Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the World, Unite!!! In *J. Grid Computing*. Springer, 2006.

[14] Walfredo Cirne, Daniel Paranhos, Francisco Brasileiro, Luís Fabrício W. Góes, and William Voorsluys. On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems. In *Parallel Computing*, volume 33. Elsevier, 2007.

[15] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proc. Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[16] Pasquale Cozza, Carlo Mastroianni, Domenico Talia, and Ian Taylor. A Super-Peer Protocol for Multiple Job Submission on a Grid. In *Proc. CoreGrid Workshop on Grid Middleware, Euro-Par*, August 2006.

[17] Fabrício A. B. da Silva, Sílvia Carvalho, and Eduardo R. Hruschka. A Scheduling Algorithm for Running Bag-of-Tasks Data Mining Applications on the Grid. In *Proc. Euro-Par*, Pisa, Italy, 2004.

[18] Xavier Dalem. Implémentation d'un Grid Peer-to-Peer. Master's thesis, University of Liège, Liège, Belgium, June 2007.

[19] Frédéric Desprez and Antoine Vernois. Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid. In *J. Grid Comp.* Springer, 2006.

[20] Ian Foster. What is the Grid ? A three Point Checklist. *Grid Today*, July 2002.

[21] Sébastien Jodogne, Cyril Briquet, and Justus Piater. Approximate Policy Iteration for Closed-Loop Learning of Visual Tasks. In *Proc. European Conference on Machine Learning*, Berlin, Germany, 2006.

[22] Ali Kaplan, Geoffrey C. Fox, and Gregor von Laszewski. GridTorrent Framework: A High-performance Data Transfer and Data Sharing Framework for Scientific Computing. In *Proc. Grid Computing Environments, Supercomputing Workshops*, Reno, NV, USA, November 2007.

[23] Simon G. M. Koo, C. S. George Lee, Karthik Kannan, and Sze-wan Kwong. On the Economics of Peer-to-Peer Content Distribution Systems for Large-volume Contents. In *Proc. SCI*, Orlando, FL, USA, July 2004.

[24] E Lawler, J. Lenstra, Kan A. Rinnooy, and D. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley Series in Discrete Mathematics & Optimization. Wiley, New York, 1985.

[25] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. Understanding BitTorrent: An Experimental Perspective. Technical report, INRIA, Sophia Antipolis, France, 2005.

[26] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra A. Hensgen, and Richard F. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Heterogeneous Computing Workshop*, San Juan, Puerto Rico, April 1999.

[27] Al-Mukaddim Khan Pathan, James Broberg, Kris Bubendorfer, Kyong Hoon Kim, and Rajkumar Buyya. An Architecture for Virtual Organization (VO)-based Effective Peering of Content Delivery Networks. In *Proc. UPGRADE-CN'07, HPDC Workshops*, Monterey Bay, CA, USA, June 2007.

[28] Arun Ramakrishnan, Gurmeet Singh, Henan Zhao, Ewa Deelman, Rizos Sakellariou, Karan Vahi, Kent Blackburn, David Meyers, and Michael Samidi. Scheduling Data Intensive Workflows Onto Storage-Constrained Distributed Resources. In *Proc. CCGrid*, Rio de Janeiro, Brazil, May 2007.

[29] Kavitha Ranganathan and Ian Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proc. HPDC*, Edinburgh, Scotland, UK, 2002.

[30] Elizeu Santos-Neto, Walfredo Cirne, Francisco Brasileiro, and Aliandro Lima. Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids. In *Proc. Workshop Job Scheduling Strategies for Parallel Processing*, New York, NY, USA, 2004.

[31] Swaminathan Sivasubramanian, Guillaume Pierre, and Maarten van Steen. Autonomic Data Placement Strategies for Update-intensive Web Applications. In *Proc. Int. Workshop Advanced Architectures and Algorithms for Internet Delivery and Applications*, Orlando, FL, USA, June 2005.

[32] D. Stutzbach, D. Zappala, and R. Rejaie. The Scalability of Swarming Peer-to-Peer Content Delivery. In *Proc. IFIP Networking*, Waterloo, Ontario, Canada, May 2005.

[33] Baohua Wei, Gilles Fedak, and Franck Cappello. Collaborative Data Distribution with BitTorrent for Computational Desktop Grids. In *Proc. ISPDC*, Lille, France, 2005.

[34] Baohua Wei, Gilles Fedak, and Franck Cappello. Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent. In *Proc. Grid*, Seattle, WA, USA, 2005.

[35] Wikipedia, the Free Encyclopedia.

[36] A. Zissimos, K. Doka, A. Chazapis, , and N. Koziris. GridTorrent: Optimizing data transfers in the Grid with collaborative sharing. In *Proc. Panhellenic Conference on Informatics*, Patras, Greece, May 2007.

Biographical Notes

Cyril Briquet is a PhD student in Computing Science and teaching assistant at the EECS Department (Montefiore Institute), University of Liege, Belgium. He holds an M.S. in Computing Science from the University of Liege, Belgium. His research interests include P2P Grid computing, scheduling, discrete-event simulation and algorithmics.

Xavier Dalem holds an M.S. in Computing Science and is a teaching assistant and researcher at the EECS Department (Montefiore Institute), University of Liege, Belgium. He is interested in P2P Grids and their integration with other P2P technologies, namely data transfer and resource discovery. Other domains of interest include web applications and the Open Source community.

Sébastien Jodogne received the M.S. and PhD degrees in Computing Science from the University of Liège, Belgium, respectively in 2001 and 2006. His research work was acknowledged by the I.B.M. Belgium Computer Science Award in 2002. Since 2007, he works as a Vision Software Engineer for Euresys, a major player in the field of machine vision offering high-performance products and services for image acquisition and vision-oriented analysis. His research interests include theoretical computer science, machine learning and computer vision.

Pierre-Arnoul de Marneffe is Professor of Algorithmics at the EECS Department (Montefiore Institute), University of Liege, Belgium. He holds a PhD in Applied Sciences from University of Liege, and a PhD in Computing Science from Cambridge University, UK. His main research domains are algorithmics and cellular automata.