Scheduling Data-Intensive Bags of Tasks in P2P Grids with BitTorrent-enabled Data Distribution

Cyril Briquet
C.Briquet@ulg.ac.be

Xavier Dalem
Xavier.Dalem@student.ulg.ac.be

Sébastien Jodogne
Jodogne@montefiore.ulg.ac.be

Pierre-Arnoul de Marneffe
PA.deMarneffe@ulg.ac.be

Department of Electrical Engineering and Computer Science
University of Liège
Montefiore Institute, B37, B-4000 Liège, Belgium

ABSTRACT

Scheduling Data-Intensive Bags of Tasks in P2P Grids leads to transfers of large input data files, which cause delays in completion times. We propose to combine several existing technologies and patterns to perform efficient data-aware scheduling: (1) use of the BitTorrent P2P file sharing protocol to transfer data, (2) data caching on computational Resources, (3) use of a data-aware Resource selection scheduling algorithm similar to Storage Affinity, (4) a new Task selection scheduling algorithm (Temporal Tasks Grouping), based on the temporally grouped scheduling of Tasks sharing input data files. Data replication is also discussed.

The proposed approach does not need an overlay network or Predictive Communications Ordering, making our operational implementation of a P2P Grid middleware easily deployable in unstructured P2P networks. Experiments show that performance gains are achieved by combining BitTorrent, caching, Storage Affinity and Temporal Tasks Grouping. This work can be summarized as combining P2P Grid computing and P2P data transfer technologies.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems* ; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*

General Terms

Algorithms, Performance

Keywords

Grid, peer-to-peer, P2P, BitTorrent, data sharing, caching, replication, bag of tasks, scheduling

UPGRADE-CN'07, June 26, 2007, Monterey, California, USA.

1. INTRODUCTION

A Peer-to-Peer (P2P) Grid [1, 2, 3, 4] is a Grid [5] in which entities are under distinct, or even unrelated control [6], and resources such as computing time are exchanged between the Peers. The lack of trust between Peers implies that no Peer has access to the internal management data of other Peers. Therefore, Peers have to model the behavior of other Peers through repeated interactions in order to reach reciprocity of resource exchanges.

Applications that can be structured as a set of independent computational Tasks are called Bags of Tasks [1, 2] (BoT). Scheduling a BoT whose Tasks share large input data (Data-Intensive BoT) is not trivial in a P2P Grid. Data transfers should be taken into account to avoid that multiple simultaneous data transfers become a bottleneck.

Several recent works [1, 7, 8, 9, 10, 11, 12] integrate data management with Task scheduling using patterns such as data replication and caching. It has also been proposed to use the BitTorrent P2P file sharing protocol [13, 14] to transfer input data of Tasks in Desktop Grids [15, 16, 17]. However, either the proposals aren't well adapted to P2P Grids, or they don't fully prevent potential bottlenecks resulting from the simultaneous multiple transfers of large data.

This paper brings together P2P Grid computing and P2P data transfer technologies. In the context of P2P Grids, we propose a novel combination of data-aware Task scheduling patterns, data caching, replication. and relying on BitTorrent for data transfers.

The rest of the paper is structured as follows: Section 2 describes the P2P Grid model, Section 3 reviews related work and motivates this work, Section 4 presents our proposal of data-aware Task scheduling in P2P Grids, Section 5 discusses the possibility of asynchronous, proactive data replication, Section 6 evaluates the performance of the proposed Task scheduling mechanism, and finally Section 7 concludes.

2. GRID MODEL

The Lightweight Bartering Grid Architecture [3] (LBG architecture) specifies a lightweight P2P Grid architecture where computing time is exchanged (with so-called bartering methods) between Peers.

Independent entities under separate administrative control are represented by software agents, called Peers. The role of Peers is to process Bags of Tasks submitted by their

users. A Bag of Tasks [1, 2] (BoT) is a set of independent computational Tasks. So-called Bags of Tasks constitute an important class of applications, covering domains such as computer vision, data mining, geographical information systems (GIS), image processing, discrete optimization, massive search (i.e. pattern matching, protein docking, ...), parameter sweeps.

Peers use their own computational Resources or Resources belonging to other Peers. In the latter case, a Peer using the Resources of another Peer is a consumer and a Peer running Tasks of another Peer is a supplier. Resources of a given Peer may be supplied (see figure 1) to execute work units of other Peers at the Task level.

In the LBG architecture, a Grid is a transient Peer-to-Peer network that emerges in a bottom-up fashion and where Peers exchange Resource usage time. Resource exchange is a way to potentially speed up BoT completion time. Indeed, synchronized consumption by a given Peer of Resources belonging to multiple Peers may dramatically reduce BoT completion times.

A Peer can be seen as a system that receives and processes computing requests (BoTs) submitted by its users or by other Peers. The main components of a Peer are a set of thread-safe, loosely coupled managers. Each Peer manages incoming BoTs, queued and scheduled Tasks, as well as the data associated with these Tasks. It also manages information about the state of its Resources, and the state of the Resources supplied by other Peers. It negotiates Resource exchange with other Peers, schedules and manages the execution of Tasks.

3. MOTIVATION AND RELATED WORK

3.1 Data-Intensive Bag of Tasks Scheduling

Data-Intensive Bags of Tasks [9, 10, 11] constitute a subclass of the Bags of Tasks applications, where the amount of processed data is large, leading to data transfer times that are long compared to computing times. These applications have also been called PHD (Processors of Huge Data [9]). With increasing capacities of data acquisition, most of the applications domains mentioned in the previous Section can be structured as Data-Intensive BoTs.

At the same time, with the development of Grid computing technologies, large-scale sharing of resources is increasingly being performed on a wider range of infrastructures, some of which are not managed at all and whose components are therefore very unreliable, i.e. home computers, unmanaged desktop computers. The goal of these technologies is to provide reliability and nontrivial [18] Quality of Service (QoS) to their users in an unreliable environment. Examples include Desktop Grids [19] and Peer-to-Peer Grids [1, 3], the former being more focused on cycle stealing within an organization, and the latter being more oriented towards collaboration between entities from separate, even unrelated control domains.

In P2P Grids, runtimes and data transfer times may be variable, and conflicts in resource usage may be the norm rather than the exception. Scheduling BoTs in such uncontrolled environments is therefore a challenge, requiring the use of techniques taking the lack of trust and reliability into account. Well-known strategies to efficiently schedule Data-Intensive BoTs include (1) **data caching** and (2) **data replication**, and to (3) **take account of data placement** when scheduling.

A recent example of synchronous data replication in a Desktop Grid, in the context of life sciences applications [11], proposes an integer programming scheduling algorithm. It is limited to a steady state context. It is therefore not applicable in the context of this paper, because of the unreliability and constant change of Peer-to-Peer environments.

It has been shown [10] that **data replication** can be performed asynchronously from Task scheduling, with simple and cost-efficient algorithms, as long as the Task scheduler is aware of data placement.

Note also that Data-Intensive BoTs implicitly require data replication, in the sense that shared input data files have to be simultaneously transferred multiple times when Tasks are scheduled to different Resources at the same time. In this paper, we introduce the Task scheduling pattern that consists in simultaneously scheduling a maximum of Tasks needing the same input data. We call this pattern *Temporal Tasks Grouping*.

However, the costs incurred by data replication can be very high, in terms of transfer times and of infrastructure overload. The overload of Peers might lead to bottlenecks due to the thrashing caused by the handling of management threads or by the saturation of network bandwidth. Another scheduling strategy is to avoid data replication, and use **data caching** instead. A proposed Task scheduling algorithm [12] taking advantage of data caching groups Tasks sharing the same input data on the same computational resources. We call this pattern *Spatial Tasks Grouping*. Task execution parallelism is of course reduced with Spatial Task Grouping.

The Storage Affinity [9] Task scheduling algorithm in the OurGrid [1] P2P Grid middleware **takes data placement into account**. It schedules Tasks first to computational resources where most of the required input data is already available, before considering other resources where data replication is required. Given the fact that it is operating in a P2P environment, Storage Affinity has to deal with the unavailability of accurate data about computing times. Task replication is proposed as a heuristic to find good Task-to-computational resource assignments. This has the side effect that any BoT *de facto* becomes Data-Intensive, in the sense that its input data has to be transferred multiple times, increasing the amount of network traffic. Storage Affinity is definitely well adapted to P2P Grids, but suffers from the high cost of multiple simultaneous data transfers that would be required by Temporal Tasks Grouping.

The previous paragraphs can be summarized as follows. (1) Temporal Tasks Grouping allows greater parallelism in Tasks scheduling, leading to lower overall BoT runtimes. However, simultaneously transferring the input data of several Tasks scheduled at the same time rapidly leads to bottlenecks. (2) Spatial Tasks Grouping entirely avoids the problem of multiple simultaneous data transfers. Of course, Tasks sharing the same input data. are not necessarily simultaneously scheduled. It is then more difficult to offer nontrivial QoS [18], and BoT runtimes are higher. (3) Activating Temporal Tasks Grouping and Storage Affinity together may enable to maintain good performance in all setups, as they are complementary. However, this requires to find a way to prevent network bottlenecks when temporally grouping Tasks.

3.2 Simultaneous Transfers of the Same Data

A possible efficient way for a Peer to transfer the same data multiple times is to use a set of data caches scattered over the P2P Grid. This leverages the bandwidth of several Peers and partially redistributes the transfer load across the P2P Grid. Recent work includes the Super-Peer model [7] and the File Mover overlay network [8]. They however both require the explicit deployment of Peer-independent data caches as well as of a routing substrate, or overlay.

Another efficient way for a Peer to transfer the same data multiple times is to use the BitTorrent [13, 14] P2P file sharing protocol. BitTorrent does not need an overlay to be constructed on top of Peers and, as such, is easy to deploy. A recent study [20] shows that BitTorrent performs nearly as well as overlay-based techniques in over-provisioned network cores, but also indicates that BitTorrent sustains *"equivalent undegraded performance"* when the available bandwidth decreases. In other words, BitTorrent has good performance in managed and over-provisioned networks and, as opposed to overlay-based techniques, maintains good performance in bandwidth-constrained networks that are more typical to P2P Grids environments.

An efficient way to greatly reduce the cost of transmitting multiple times the same data to different resources in a Desktop Grid has been recently and independently proposed in two studies [15, 16]. The idea is to use BitTorrent [13, 14] to distribute Tasks data throughout the Grid.

Let us briefly discuss how BitTorrent works. A Peer, called *a seeder*, that wants to share a file with BitTorrent first starts by splitting it into pieces. It then launches a *tracker*, or may use a publicly available tracker, to which it invites Peers to connect to get introduced to one another. Each Peer initially downloads a first piece from any of the Peer communicated by the tracker, and then begins to exchange pieces with these other Peers. A Peer invites other Peers to collaborate by uploading pieces to them. With BitTorrent, as opposed to what happens with direct file transfers protocols, network links between Peers are exploited (see figure 2): As each downloader is also an uploader, the network load is removed from the seeder and distributed to all Peers. As opposed to other P2P file sharing protocols, Peers using BitTorrent do not have to wait for a file transfer to be completed to begin uploading pieces of it to other Peers. Indeed, BitTorrent enables Peers to simultaneously act as downloaders and uploaders as soon as they begin to download a file, while allowing them to continue to act as uploaders when the file transfer has been completed.

In the context of this paper, an advantage of BitTorrent over regular File Transfer Protocol (FTP) is that the total transfer time of a file by multiple downloaders is not linearly dependent on, and increases remarkably slowly with, their number. Coupling Grid scheduling and BitTorrent data transfer therefore offers a very interesting perspective: As more Tasks sharing input data 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. This enables to use Temporal Tasks Grouping, without excluding the use of Spatial Tasks Grouping.

In the first mentioned study about Grid scheduling relying on BitTorrent [15], a model of BitTorrent transfer times is proposed (building upon previous work [17]), as well as several BitTorrent-aware versions of classic, knowledge-based scheduling heuristics (BT-MinMin, BT-MaxMin, BT-Sufferage). This very interesting work is, to the best of our knowledge, the first proposal of coupling Grid scheduling and BitTorrent data transfer. The BT-X knowledge-based scheduling heuristics [15] 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 specifically not to be trusted in a P2P environment. Furthermore, the proposed heuristics use a modified version of BitTorrent, whose loose reciprocity-based policy (i.e. the choking algorithm, whose operation may be described as enforcing loose reciprocity) has been replaced by Predictive Communications Ordering (PCO). This is only possible when Peers downloading input data can be controlled, 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 this paper.

In the second mentioned study [16], a Computer Vision Learning problem, structured as a Data-Intensive Bag of Tasks application, is presented and shown to be successfully computed with a non-dedicated, proprietary Desktop Grid. In this context, BitTorrent is used to simultaneously transfer half-gigabyte-sized data to several dozens computing resources.

To complete this review of related work, we explain why GridFTP [21, 22] is not an appropriate technology in the context of this paper, i.e. simultaneously transferring the same data file to multiple Peers in a P2P Grid. 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 data transfers, i.e. parallel transfers of a file through several network interfaces. Clearly, GridFTP-based striping cannot be used in the P2P setup considered in this paper, where Resources are unmanaged, not high end, and rarely with more than one network interface or huge network bandwidth. More importantly, GridFTP does not exploit the network links between downloaders, which is the key strength of BitTorrent as considered in this paper. However, as Allcock et al. [22] 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 Peers.

4. DATA-AWARE TASK SCHEDULING

We propose to combine several existing patterns to perform efficient data-aware scheduling of Bags of Tasks in P2P Grids: (1) use of the BitTorrent P2P file sharing protocol to transfer data (see figure 2), (2) data caching on computational resources, (3) use of a data-aware Resource selection algorithm similar to Storage Affinity, (4) a novel data-aware Task selection scheduling algorithm, Temporal Tasks Grouping, based on the temporally grouped scheduling of Tasks sharing input data files.

The rest of this Section is as follows: Section 4.1 introduces data management and explains how data is managed by Resources. Building on the available support for data management and transfer, Section 4.2 presents the proposed Data-Aware Task scheduling mechanism, first by explaining Task selection (Temporal Tasks Grouping) and then Storage Affinity-like Resource selection (Spatial Task Grouping).

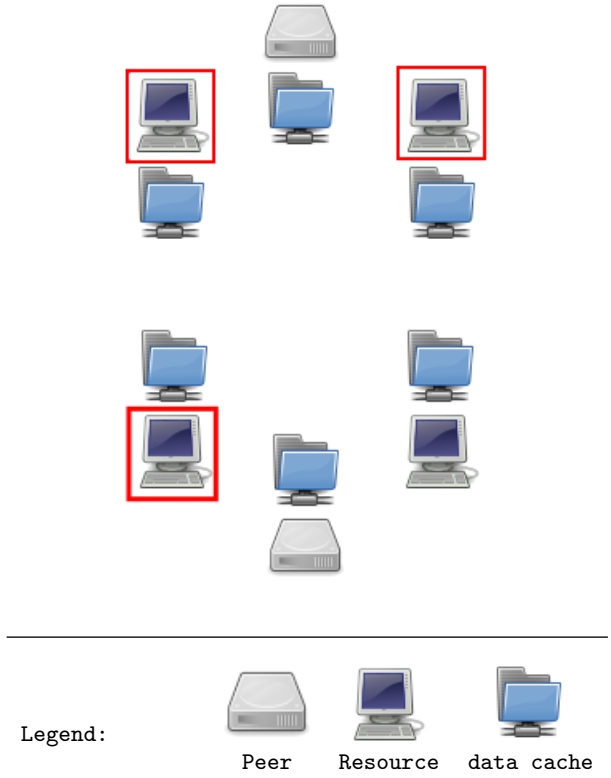

Figure 1: A Peer in consumer role (top) uses its 2 Resources, as well as 1 Resource supplied by another Peer (bottom) in supplier role.

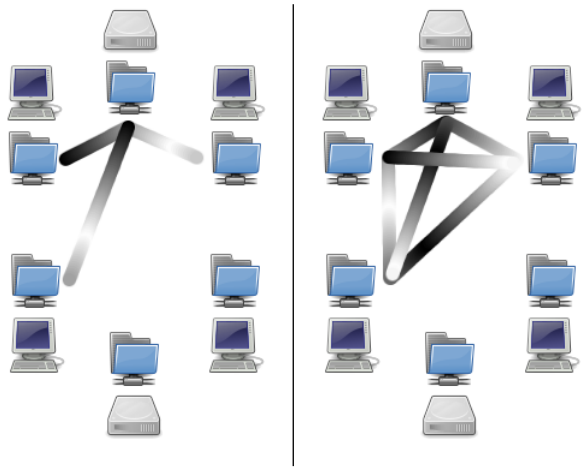

Figure 2: A Peer shares a file with several Resources. FTP data sharing (left) - only network links with the Peer are exploited. BitTorrent data sharing (right) - network links between all downloaders are also exploited, leading to file download times essentially independent of the number of downloaders.

4.1 Data Management

4.1.1 Data Managers

The data transfer mechanism can be simply described as follows: Input data files of a given Task are transferred synchronously, at Task submission time, to the Resource to which this Task is submitted. Data transfers may be performed with either BitTorrent or with FTP, given the context and scheduling decisions, as will be explained in Section 4.2.2. The timing of data transfers, or data scheduling, depends on the timing of Tasks execution, or Tasks scheduling, which will be explained in Section 4.2.

Data transfers are managed by components called Data Managers. One Data Manager equips each Peer and each Resource. A Peer Data Manager manages the input data files of queued Tasks, while a Resource Data Manager manages downloaded input data files.

Each Data Manager has several responsibilities: (1) data storage (on both Peers and Resources), (2) BitTorrent data sharing (tracking and seeding on Peers in consumer role, and seeding only on Resources) and FTP data sharing (on Peers in consumer role only), and (3) BitTorrent and FTP data downloading (Resource only). As can be seen, responsibilities of a Peer Data Manager and of a Resource Data Manager overlap but are not equal. In other words, each Peer hosts a BitTorrent tracker, a BitTorrent client, and an FTP server, and each Resource hosts a BitTorrent client and an FTP client. Note that all these data clients and servers are embedded within the implemented middleware, not requiring extra support from the underlying systems.

A simple file naming scheme is used to provide Grid-wide naming unicity (e.g. peer_name.user_name.file_name). Metadata information is associated with each file, including Grid filename combined either with BitTorrent torrent metadata or an FTP URL.

When an input data file has to be transferred from a Peer (in consumer role) to a Resource of another Peer (in supplier role), the file metadata is first sent to the other Peer, which forwards it to its Resource. It is the Resource that actually initiates and controls the file download (similarly to the Super-Peer model [7]). If FTP is selected, the file is directly downloaded from the Peer sharing it. If BitTorrent is selected, the file is simultaneously downloaded and shared with other Peers and Resources already sharing or downloading it. The data path between a consumer Peer and a Resource may therefore not be direct, and the Resource may download an input data file entirely from other Resources which have already downloaded (parts of) it. As long as an input data file is not cleared from data storage of a Resource or Peer, it remains available for download.

4.1.2 Resource Data Management

Each Resource Data Manager is equipped with a data cache that manages the storage of data files. The only supported operation on a Resource data cache is synchronization with a working set communicated by the Peer that owns the Resource. A Resource data cache is controlled by 3 parameters: (1) a cache size, (2) a working set and (3) a cache replacement policy.

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

The working set is the set of files that the data cache should have in storage. It is communicated by the Peer

regularly, as well as each time a Task is being run. Each item in the working set is the metadata of a file, containing a BitTorrent torrent metadata or a FTP URL as explained above.

When a file in the working set is not present in the data cache, the Resource downloads it from its owner Peer or from another consumer Peer in the P2P Grid, depending upon the origin of the Task using this file. The Resource may also very well download it from other Peers when BitTorrent is used.

A consumer Peer always includes in the working set the files needed by the Task to run, or by the Task currently being run on the Resource to which it is scheduled. The Peer always guarantees that the working set can always be fully stored in the cache: The Peer never schedules a Task to a Resource with insufficient cache size.

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. It means that the least recently used (and not in the working set) files are ejected from the cache.

4.2 Task Scheduling

The performance of a set of BitTorrent data transfers is better when they happen simultaneously, as opposed to FTP data transfers. As data transfer scheduling depends on Task scheduling, the goal of the proposed scheduling algorithms is to schedule at the same time Tasks sharing the same input data files.

4.2.1 Task Selection

Let us now discuss in which order the Tasks are scheduled. Let $\theta = \theta_0, \ldots, \theta_{n-1}$ be a Bag of Tasks. Let Δ_i be the set of input data files (simply called data in the following) 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 data 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*. Note that 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 two input data sets of tasks θ_i, θ_k is the sum of the sizes of the data of θ_k that are not shared 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 data, 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 [23] for each *sequence*. Instead, we propose the **Temporal Tasks Grouping** (TTG) algorithm to schedule Tasks sharing data at the same time. TTG deals with a *sequence* of Tasks, and is applied to all *sequences* of a BoT, which are then sorted.

To explain the Temporal Tasks Grouping algorithm, let us introduce the following definitions. Two Tasks are said to be *data-equal* when they have all their data 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. We propose to sort the *data-equal subsequences* of a *sequence* by decreasing length $|\bar{\sigma_s}(\theta)|$ (see figure 3).

Original Tasks input data files of Bag of Tasks θ

Δ_0	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7
$\{g\}$	$\{i\}$	$\{g\}$	$\{r\}$	$\{g\}$	$\{g\}$	$\{d\}$	$\{r\}$

Sorted Tasks input data files of Bag of Tasks θ

Δ_0	Δ_2	Δ_4	Δ_5	Δ_3	Δ_7	Δ_1	Δ_6
$\{g\}$	$\{g\}$	$\{g\}$	$\{g\}$	$\{r\}$	$\{r\}$	$\{i\}$	$\{d\}$

Figure 3: Tasks of Bag of Tasks θ (here with 1 data file per Task) are grouped into *data-equal subsequences*, which are in turn sorted by decreasing length $|\bar{\sigma_s}(\theta)|$.

TTG therefore ensures that the scheduling of *data-equal* Tasks is temporally grouped, so as to maximize efficiency of BitTorrent transfers. It also ensures that the largest groups of *data-equal Tasks* are scheduled first, at a time when a large number of suppliers is supposed to be available.

Multiple *data-equal subsequences* of similar length may then be sorted using a nearest neighbour algorithm [23] (the metric being the distance $d(\Delta_i, \Delta_k)$ between Tasks). Importantly, this ordering can be performed at the submission time of the BoT. Task selection is simple and consists in following the precomputed ordering.

4.2.2 Resource Selection

Data placement, or **Spatial Task Grouping**, is explicitly taken into account when selecting a Resource to schedule a Task locally and implicitly when a Task is scheduled to a supplier Peer, which might have one Resource storing the required data in its cache. Let Δ_{R_x} be the contents of the data cache of Resource R_x. At a given time, it contains data accumulated from previous Task executions. When a Peer is scheduling Task θ_i locally, a Resource is located by minimizing the distance $d(\Delta_i, \Delta_{R_x})$ between the data cache of each Resource and the input data set Δ_i of the Task to schedule. This distance, computed dynamically, represents the transfer cost of scheduling θ_i on R_x and requires data tracking support. The minimum distance will usually be small, as there will probably be one Resource whose data cache already stores most of data of Δ_i due to the execution of previous Tasks. This minimization is equivalent to maximizing Storage Affinity [12].

Finally, as it would not be efficient to share the input data of Tasks with data-equal subsequences that are short either in length or number, a Peer should share some of its data with FTP rather than BitTorrent. For this decision, we rely on recent related work proposing a simple analytical time model of multiple simultaneous transfers of a given file through BitTorrent [15, 17]. The model is a function of the number of downloaders, file size, server bandwidth, protocol latency, and involves a factor of the number of downloaders that is logarithmic.

5. DATA REPLICATION

5.1 Replication May Increase Reputation

5.1.1 Reactive, Synchronous Data Transfers

The objective of the Temporal Tasks Grouping algorithm is that consumer Peers simultaneously schedule as many Tasks of a BoT sharing input data files as possible, on multiple supplier Peers, preferably where these input data files are already available. Resources (from a supplier Peer that is going to run a Task) first download the required input data if it is not already present in their data cache. This algorithm requires very little management data, and especially none from other Peers. Its main benefit is that it maximizes parallelization of Tasks execution and is compatible with data-aware Task scheduling that can avoid unnecessary data transfers [9, 12]. In a P2P Grid, available Resources of a Peer may vary during the execution of a BoT. Another benefit of Temporal Tasks Grouping is that it is adaptive as it can opportunistically makge good use of extra supplementary Resources as they become available by scheduling Tasks sharing data with already scheduled Tasks.

To use Temporal Tasks Grouping, BitTorrent support [15, 16, 17] is mandatory in order to control the cost of multiple simultaneous transfers of the same large data. As explained in Section 4.1, data transfers are performed synchronously when input data files needed by a given Task are not present in the data cache of the Resource where the Task has been scheduled. In this case, data scheduling simply depends on Task scheduling.

5.1.2 Proactive, Asynchronous Data Transfers

In order to increase its reputation as a supplier, a Peer could ask its Resources to download extra copies of a popular input data file that has a good chance of being required in a near future by Tasks from another Peer. If this input data file is actually required by some future Supplying Tasks, it will already be available, thus decreasing the response time. The rationale is based on the well-known locality principle: If several Tasks (of a given consumer Peer) sharing the same input data have been run in a recent past, more of them will be run in a near future, at nearby locations.

To achieve this, we propose that supplier Peers proactively and asynchronously command some of their idle Resources to download input data, using BitTorrent, either from themselves or from other consumer Peers. The cost of doing so is low due to the joint use of Temporal Tasks Grouping and BitTorrent, which enable extra transfers at low cost. Proactive, asynchronous data transfers are independent from reactive, synchronous data transfers but happen concurrently so as to benefit from the joint use of BitTorrent and Temporal Tasks Grouping.

5.2 Asynchronous Data Scheduling

Our proposal of proactive, asynchronous data transfers is similar to, yet slightly different from, previous related state of the art work [10] in the sense that it must be *pull-based* rather than *push-based*. Suppliers, rather than consumers, initiate data replication for two reasons. (1) Firstly, lack of trust in P2P Grids precludes that a consumer Peer pushes data to supplier Peers independently of a Task to run (i.e. without control of reciprocity of actions). This limitation is important because it removes a possible vector for *Denial of Storage* attacks (i.e. a malicious Peer uploads data to another Peer until its storage resources are exhausted). (2) Secondly, supplier Peers have a double interest in proactively downloading data from probable consumer Peers. (2a) Supplying a lot of Resources to few consumers is better (bartering reputation-wise) than supplying few Resources to a lot of consumers. And, as the goal of consumer Peers bartering with one another, it is intrinsically good for a consumer to gather instantaneously as many Resources as possible. (2b) The cost of doing so can be controlled bandwidth-wise, thanks to BitTorrent's efficient handling of flash crowds of downloaders, and storage-wise, thanks to the fact that the Peer can limit the amount of storage involved as it chooses to initiate the proactive data transfers.

As explained in Section 4.1, each Resource is equipped with a data cache. The content of a data cache is controlled by its working set, which is communicated by the Peer owning the Resource. The working set of a Resource may be updated synchronously with Task scheduling, and also asynchronously. A working set communicated synchronously always includes the input data file of the scheduled Task. A working set communicated asynchronously should include input data files that are predicted to be required in a near future by Tasks soon to be scheduled. A key decision is therefore to decide which input data files will probably be required by the owner Peer, or in other words, which ones new Supplying Tasks will depend on.

Ranganathan proposed a popularity metric [10] that is defined as the number of times an input data file is required as input to scheduled Tasks. To take into account popularity recency, popularity tracking will be limited to the K most recent Tasks submitted to the Peer, using a sliding window technique.

In a spirit similar to BitTorrent's optimistic unchoking policy [13, 14], adding some randomness will augment the diversity of the replicated data. Replicating not only the most popular input data files but also a small number of randomly selected input data files will increase the probability of moderately popular data to be replicated. We propose to fill the working set of a given Resource with two distinct sets: the most popular data (intrinsic high utility), and some random data (utility diversity) among the K most recent scheduled Supplying Tasks. These two sets are assigned weights of 0.8 and 0.2, respectively. Note that replicating input data files because they are popular is also well aligned with the use of BitTorrent in the sense that popular files will be stored on many Resources, thus providing a large number of potential download sources and a more balanced network load.

To summarize, there are two forms of data scheduling that may happen concurrently: (1) reactive, synchronous data transfers that may occur each time a Task is scheduled to a Resource, if input data files are not already present in the data cache of this Resource, and (2) proactive, asynchronous data transfers that may occur when a Peer decides to replicate popular input data files to idle Resources. Efficient data replication requires that BitTorrent and Temporal Tasks Grouping are jointly used. Selecting popular input data files for replication is efficient for two reasons: (1) because of the locality principle, and (2) because it is making full use of BitTorrent's features.

6. EXPERIMENTS

6.1 Environment

The proposed Task selection scheduling algorithm (Temporal Tasks Grouping), the Resource selection scheduling algorithm minimizing data transfers (Storage Affinity), caching support, both modes of data transfers (synchronous, reactive, and asynchronous, proactive) have been implemented in the Java language (J2SE 5.0 [24]), and BitTorrent and FTP support are embedded thanks to the use of Java libraries.

This implementation is deeply integrated with the Lightweight Bartering Grid middleware [3], a recent P2P Grid middleware similar to OurGrid [1, 2]. The BitTorrent tracker and client, as well as the FTP server and client are off-the-shelf versions of, respectively, Azureus [25] (version 2.5.0.4), Apache FTP server [26] (version 20061027, slightly patched to enhance security by denying several FTP commands), and edtFTPj [27] (version 1.5.3), which are all widely available implementations of the BitTorrent and FTP protocols. This 100% Java implementation is therefore easily deployable as-is on a great number of platforms.

The algorithms presented in this paper have been evaluated on a cluster of 28 ix86 PC (Intel P-IV 3GHz with 1Gb RAM), all equipped with commodity hard drives, and connected with switched 100Mbps Ethernet. Any PC of this cluster may be used as a Resource or a Peer.

6.2 Deployment

In a P2P Grid, each Peer hosts its own data server, as opposed to a Desktop Grid, that must rely on a centralised data server. Each Peer runs both 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 users. Each Resource runs a BitTorrent client and an FTP client, and offers data caching support. Thus, each Resource contributes to data transfers within the P2P Grid, as it acts as a transient data server to other Resources by participating to BitTorrent downloads.

A P2P Grid with several Peers, one acting as a consumer and one or more acting as suppliers, is deployed. The consumer Peer does not own any Resource, as the focus of this paper is on data transfers required by Tasks executed on Resources of multiple Peers. Note that Tasks executed on Resources controlled by the Peer to which they were submitted simply constitute a special case of Tasks executed on Resources of multiple Peers, as far as the data transfers are concerned. A simulated user submits Bags of Tasks to the consumer, but not to the suppliers. One 256 MB input data file is associated to each Task. The activity of each Task consists in simply pausing for 2 seconds and then hashing its associated input data file, so as to minimize and control the impact of Task runtimes on the overall runtime of Bags of Tasks.

The local scheduling policy used by Peers is FIFO with limited preemption (and requeueing) of Tasks from other Peers as needed, in order to give priority to incoming local Tasks. To avoid unnecessary preemption, Tasks from other Peers are scheduled only when there is no queued local Tasks. The Peer supplying policy is based on Favors ranking, following the Network of Favors model [2]. Therefore, a Peer without queued local Tasks always accepts incoming Tasks from other Peers, in order to promote its ranking towards other Peers and contribute to the sustainability of the P2P Grid. Note that in order to avoid interferences arising from queueing issues in the presented experiments, the deployed consumer has no Resources, and the deployed suppliers are not submitted any local Tasks, but instead are asked by the consumer to process some of its Tasks.

6.3 Results

6.3.1 Scenario 1

The following scenario is investigated: There is one consumer Peer with no Resources and 3 supplier Peers owning 8 Resources each. A user submits Bags of 100 Tasks to the consumer Peer.

Experiments are controlled with five parameters. (1) Task selection scheduling using Temporal Tasks Grouping (TTG), which is done statically. (2) Resource selection scheduling of the Resource minimizing data transfers, i.e. with highest Storage Affinity (SA), which is done dynamically. (3) Data transfer protocol (BitTorrent or FTP). (4) The level of sharing (i.e. redundancy) of input data in a BoT is taken into account. A measure of the sharing of input data between Tasks of a BoT, the Shared Data Ratio (SDR), is used. The SDR is defined as the size of the unique input data files among all Tasks of the BoT, divided by the total size of all input data files of the Tasks. If there is no sharing of data, SDR = 1.0. The SDR decreases towards 0 as the sharing increases ($0 < \text{SDR} \leq 1.0$). In this scenario, a medium SDR (0.25) is used, to simulate BoTs with some data redundancy, so as to be representative of real-world Bags of Tasks. (5) Resource cache size, i.e. the number of input data files that can be stored on a Resource.

Two metrics are measured: BoT response time and Peer cache hit ratio. BoT response time is the elapsed time (in seconds) between BoT submission time to a Peer and the time when results of all Tasks have been uploaded back to the user. Users typically expect low BoT response times. The mean cache hit ratio of a Peer (a real number between 0.0 and 1.0) is the mean of the cache hit ratios of Resources of a given Peer. Peer administrators typically expect high cache hit ratios.

Figure 4 shows 3 sets of measured BoT response times. Each set includes response times for a cache with a capacity of 0, 1 and 15 files, respectively. The 3 sets correspond to, respectively, { FTP + SA + TTG }, { BitTorent + SA + TTG } and { BitTorrent + SA + no TTG }.

The cache hit ratio converges around 0.75 for all experiments with caching support (i.e. cache size > 0), except when BitTorrent and Temporal Task Grouping are both activated. In this situation where BitTorrent and TTG are activated, there is indeed little opportunity for cache hits, as Tasks requiring the same input data files are scheduled simultaneously on different Resources.

A first conclusion is that, as expected, BitTorrent transfers clearly outperform FTP transfers in all but one setup (FTP with large cache capacity). As BitTorrent has a larger overhead than FTP, its advantage over FTP grows as the transfers become larger. Therefore, when cache capacity is large (15 files), FTP transfers happen only once for each file as the cache hit ratio is very high. With large cache capacity, which reduce the amount of transfers to be actually performed, BitTorrent overhead dominates the BoT response time, which becomes a little longer than what is

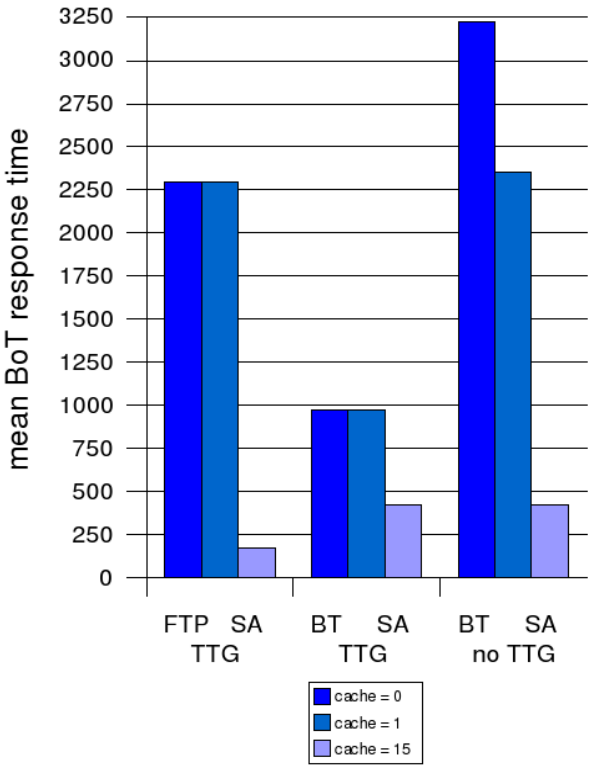

Figure 4: Mean BoT response time (in seconds)

achieved with FTP transfers.

A second conclusion is that caching has a huge impact. BitTorrent without support from Temporal Task Grouping benefits proportionally the most from data caching, probably because even when a cache miss happens on a given Resource, other Resources act as uploaders when their cache holds the expected data.

A third conclusion is that Temporal Task Grouping enables the full power of BitTorrent, which leads to small response times even with very low levels of caching. It is interesting to note that allowing the deployment of Resources with limited cache capacities has a strong practical interest, as files may be very large.

6.3.2 *Scenario 2*

The following scenario is investigated: There is one consumer Peer with no Resources and 1 supplier Peer owning 8 Resources. A user submits 8 times the same Bag of 8 Tasks to the consumer Peer. The 8 Tasks are associated with a different 256 MB input data file (SDR = 1.0), and stored in random order within the BoT.

Transfers are performed with BitTorrent. Resource selection with highest Storage Affinity is activated and then deactivated, and a cache size of 1 is used. Given the small cache size, it is important to correctly schedule Tasks on Resources that already store the required data.

In this controlled setup showing inter-BoT data sharing (sometimes referred to as inter-Job data sharing [9]), the cache hit ratio is very high (0.88) in both cases. Resource selection with highest Storage Affinity avoids most data transfers for all BoTs following the first submitted BoT, decreasing the mean BoT response time from 242s to 116s.

6.3.3 *Scenario 3*

Finally, the following scenario is investigated: There is one consumer Peer with no Resources and 1 supplier Peer owning 8 Resources. A user submits Bags of 100 Tasks to the consumer Peer. The 100 Tasks have the same 256 MB input data file. The SDR (= 0.01) of such Bags of Tasks is very low, as is typically the case for so-called parameter sweeps applications.

Transfers are performed with BitTorrent. Temporal Tasks Grouping and selection of Resource with highest Storage Affinity are enabled. The mean BoT response time is 2319s when caching is deactivated. It decreases sharply to 201s with a cache size of 1, with little variations for a larger cache size. The theoretical minimum BoT response time for a setup with FTP transfers and Resource caching support would be $\sim$ 205s. The observed response time in this FTP setup is actually 303s, which is good but still slower than in the BitTorrent setup. As can be seen, BitTorrent transfers coupled with caching support on Resources is very efficient even with a small number of downloaders, i.e. 8 downloaders.

6.4 Discussion

BitTorrent data transfers, data caching support on Resources, and Task scheduling with Temporal Grouping all have an important impact on BoT response times, and gain from being combined.

BitTorrent brings important benefits in most setups, except where its overhead dominates the response time, i.e. when there is not much data to transfer.

Caching of course brings important benefits too. Determining optimal cache capacity will be explored in future work. It is however already clear that a large cache capacity brings better benefits. But when BitTorrent is used without Temporal Task Grouping support, even a very limited cache capacity brings some benefits. Alternatively, when the effect of Temporal Task Grouping is limited, for example in setups with high SDR (e.g. all files within a BoT are different), even limited small caching support is very imporant.

The effect of Storage Affinity is to ensure that, when scheduling a given Task, a Peer selects one of the Resources that have the input data files of this Task stored in its data caches. Deactivating Storage Affinity would severely degrade response time performance and cache hit ratio when scheduling Bags of Tasks with large input data files exhibiting a high SDR.

The effect of Temporal Tasks Grouping is to group the scheduling of Tasks so as to maximize the number of simultaneous downloaders of input data files. This allows BitTorrent to produce maximum benefits as it scales really well with the number of simultaneous transfers of the same file.

BitTorrent, caching and Temporal Tasks Grouping must all be activated to enable data replication to bring extra benefits. Further experiments will be required to measure how much benefits can be achieved with the proposed data replication process based on pull-based, proactive, asynchronous data transfers. Note that the cost of limited data replication is very low thanks to the scalability of BitTorrent and the maximization of simultaneous downloads of input data files created by Temporal Tasks Grouping.

Figure 5 summarizes the dependencies that exist between the technologies presented in this paper. BitTorrent and caching support must be activated for the other technologies

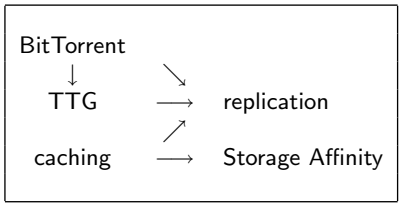

Figure 5: Technology dependencies

to have a favorable impact.

We recommend without restriction that BitTorrent and caching support are systematically activated in P2P Grids. Storage Affinity can be activated without restriction in almost all setups. On Peers where Resources exhibit large variation in computing power, Storage Affinity should also be combined with a performance-aware Resource selection algorithm. Alternatively, Storage Affinity may be combined with Task Replication [9], that relies on redundancy of Task execution as an efficient but somewhat costly way to compensate for unfortunate Tasks/Resources assignments. Alternatively again, Storage Affinity may be combined with the learning of reliability models of Resource supplying [28], as a way to avoid unfortunate Tasks/Resources assignments in the first place, rather than have to compensate for them. We also recommend that Temporal Tasks Grouping be activated in almost all setups, and should be combined with a Task selection algorithm aware of expected Peer performance when Peers in the P2P Grid show large variation in performance. Temporal Tasks Grouping and Storage Affinity are complementary: Because the benefits of caching are very high, Temporal Tasks Grouping should be activated when there is no caching support, as Storage Affinity requires caching support to be of any use. And if Temporal Task Grouping is deactivated, caching support, even very limited, should definitely be enabled. The proposed pull-based data replication should be activated only when Temporal Tasks Grouping is expected to perform well, and the tuning of its parameters require further study.

Finally, it should be noted that the consumer Peers studied in the presented experiments always consume Resources from the same set of Peers. In a general setup, the neighbourhood of a given Peer may vary over time. Therefore, a consumer Peer should seek to schedule on the same Peers any Task that would require the same input data files as Tasks previously scheduled some time before, provided that caching support is activated on the Resources of these Peers. Simultaneously scheduling on multiple Peers several Tasks requiring the same input data files remains efficient indeed, thanks to the activation of BitTorrent.

7. CONCLUSIONS

This work can be summarized as bringing together P2P Grid computing and P2P data transfer technologies. It synthesizes several previous works in Grid Task scheduling (scheduling Tasks sharing data distributed with BitTorrent [1, 3], push-based asynchronous proactive data replication [10]) and P2P Grid Task scheduling (Storage Affinity algorithm [9] and Super-Peer model [7]).

Several technologies and patterns are combined and integrated to provide efficient scheduling of Data-Intensive Bags of Tasks in P2P Grids: (1) systematic use of the BitTorrent P2P file sharing protocol to transfer large shared data, so as to prevent network bottlenecks, (2) Resource caching support that avoids transfer of data recently downloaded by the Resource, (3) a Resource selection scheduling algorithm that minimizes the amount of input data to be transferred when scheduling a given Task, (4) a new Task selection scheduling algorithm (Temporal Tasks Grouping) targeting Data-Intensive Bags of Tasks in P2P Grids, and based on the temporally grouped scheduling of data-equal Tasks of a Bag of Tasks by consumer Peers, (5) pull-based, proactive, asynchronous replication of consumer Peers' data by supplier Peers.

Experiments show that combining BitTorrent data transfers, Task selection scheduling with Temporal Task Grouping, and Resource caching support is very efficient in a P2P Grid for all setups, even with very small cache sizes, for both low and high data sharing within Bags of Tasks, and including in the special, important case of parameter sweeps.

Adding the use of a data-aware Resource selection algorithm minimizing data transfers (Storage Affinity) is important as it optimizes the cache hit ratio, but should be integrated with an algorithm taking into account the computing power of Resources in case of large performance variation of the Resources of the managed Peer.

From a deployment perspective, BitTorrent support, needed on each Peer and Resource, is embedded within the implemented middleware. Our proposed solution is easy to deploy in unstructured P2P networks, as BitTorrent does not rely on an overlay network and Predictive Communications Ordering is not needed.

Our implementation of an operational middleware demonstrates that off-the-shelf BitTorrent support can be deeply integrated with a P2P Grid middleware and brings considerable performance gains.

As future work, it will interesting to evaluate the application of Temporal Tasks Grouping to large Bags of Tasks. We also intend to explore various combinations of cache capacity, SDR, and network bandwidth. It will also be interesting to evaluate pull-based, proactive, asynchronous data replication: in particular, evaluating when it is optimal to trigger proactive replication and what is the best policy for assigning weights to the sets of replicated data (popular and random data). Future work could also explore the impact of heuristics taking Task runtimes into account when scheduling. For example, a version of the MinMin or MaxMin [17] heuristics modified to use predicted, rather than communicated, knowledge as runtime data.

Acknowledgement

We want to thank Claire Kopacz and Elisabeth Spilman for their help in the preparation of this manuscript. We also want to thank the anonymous reviewers for their constructive comments. Figures 1 and 2 include icons from the Tango library (`http://tango.freedesktop.org/`), under Creative Commons Attribution Share-Alike license.

8. REFERENCES

[1] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray, "Labs of the World, Unite!!!" in *J. Grid Computing*. Springer, 2006.

[2] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing," in *Proc. Workshop on Job Scheduling Strategies for Parallel Processing*, Seattle, WA, USA, 2003.

[3] C. Briquet and P.-A. de Marneffe, "Description of a Lightweight Bartering Grid Architecture," in *Proc. Cracow Grid Workshop*, Cracow, Poland, 2006.

[4] N. Drost, R. V. van Nieuwpoort, and H. E. Bal, "Simple Locality-Aware Co-allocation in Peer-to-Peer Supercomputing," in *Proc. GP2P*, Singapore, May 2006.

[5] F. Berman, A. J. G. Hey, and G. Fox, Eds., *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, April 2003.

[6] C. Briquet and P.-A. de Marneffe, "What is the Grid ? Tentative Definitions Beyond Resource Coordination," University of Liège, Liège, Belgium, Tech. Rep., 2006.

[7] P. Cozza, C. Mastroianni, D. Talia, and I. Taylor, "A Super-Peer Protocol for Multiple Job Submission on a Grid," in *Proc. CoreGrid Workshop on Grid Middleware, Euro-Par*, August 2006.

[8] C. Anglano and M. Canonico, "The File Mover: An Efficient Data Transfer System for Grid Applications." in *Proc. CCGrid*, Chicago, IL, USA, 2004.

[9] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. 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.

[10] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," in *Proc. HPDC*, Edinburgh, Scotland, 2002.

[11] F. Desprez and A. Vernois, "Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid," in *J. Grid Comp.* Springer, 2006.

[12] F. A. B. da Silva, S. Carvalho, and E. R. Hruschka, "A Scheduling Algorithm for Running Bag-of-Tasks Data Mining Applications on the Grid," in *Proc. Euro-Par*, Pisa, Italy, 2004.

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

[14] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Understanding Bittorrent: An experimental perspective," INRIA, Sophia Antipolis, France, Tech. Rep., 2005.

[15] B. Wei, G. Fedak, and F. Cappello, "Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent," in *Proc. Grid*, Seattle, WA, USA, 2005.

[16] S. Jodogne, C. Briquet, and J. Piater, "Approximate Policy Iteration for Closed-Loop Learning of Visual Tasks," in *Proc. European Conference on Machine Learning*, Berlin, Germany, 2006.

[17] B. Wei, G. Fedak, and F. Cappello, "Collaborative Data Distribution with BitTorrent for Computational Desktop Grids," in *Proc. ISPDC*, Lille, France, 2005.

[18] I. Foster, "What is the Grid ? a three Point Checklist," *Grid Today*, July 2002.

[19] R. Olejnik, B. Toursel, M. Tudruj, and E. Laskowski, "DG-ADAJ: a Java Computing Platform for Desktop Grid," in *Proc. Cracow Grid Workshop*, Cracow, Poland, 2005.

[20] S. Al Kiswany and M. Ripeanu, "A Simulation Study of Data Distribution Strategies for Large-scale Scientific Data Collaborations," in *Proc. CCECE*, Vancouver, BC, Canada, April 2007.

[21] "GridFTP." [Online]. Available: http://www.globus.org/grid_software/data/

[22] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus Striped GridFTP Framework and Server," in *Proc. SC*, Seattle, WA, USA, 2005.

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

[24] "J2SE 5.0 API Specification." [Online]. Available: http://java.sun.com/j2se/1.5.0/docs/api/

[25] "Azureus." [Online]. Available: http://azureus.sourceforge.net/

[26] "Apache ftp server." [Online]. Available: http://incubator.apache.org/ftpserver/

[27] "edtFTPj." [Online]. Available: http://www.enterprisedt.com/

[28] C. Briquet and P.-A. de Marneffe, "Learning Reliability Models of Grid Resource Supplying," in *Proc. Cracow Grid Workshop*, Cracow, Poland, 2005.