

Collaborative Data Distribution with BitTorrent for Computational Desktop Grids

Baohua Wei, Gilles Fedak and Franck Cappello
Laboratoire de Recherche en Informatique/INRIA Futurs
Bat 490, Université Paris Sud, 91405 ORSAY Cedex, FRANCE
Corresponding author: fedak@lri.fr

Abstract—Data-centric applications are still a challenging issue for Large Scale Distributed Computing Systems. The emergence of new protocols and softwares for collaborative content distribution over Internet offers a new opportunity for efficient and fast delivery of high volume of data. This paper presents an evaluation of the BitTorrent protocol for Computational Desktop Grids. We first present a prototype of a generic subsystem dedicated to data management and designed to serve as a building block for any Desktop Grid System. Based on this prototype we conduct experimentations to evaluate the potential of BitTorrent compared to a classical approach based on FTP data server. The preliminary results obtained with a 65-nodes cluster measure the basic characteristics of BitTorrent in terms of latency and bandwidth and evaluate the scalability of BitTorrent for the delivery of large input files. Moreover, we show that BitTorrent has a considerable latency overhead compared to FTP but clearly outperforms FTP when distributing large files or files to a high number of nodes. Tests on a synthetic application show that BitTorrent significantly increases the communication/computation ratio of the applications eligible to run on a Desktop Grid System.

I. INTRODUCTION

For the last years, the idea of High Throughput Computing over large sets of idle Desktop Computers has become more and more popular. This acceptance is partly due to the success of mainstream projects like SETI@Home [1] or Distributed.net [2] which have gathered a tremendous amount of computing power (for instance SETI@Home is claiming more than 100 TFlops as of January 2005), and the availability of generic software platforms, being open source like BOINC [3], XtremWeb [4], OurGrid [5], or commercial like Entropia [6] or Sun Grid Engine [7]. Nonetheless, Desktop Grids Systems are still restricted to a few classes of applications: mainly the embarrassingly parallel applications (bag of tasks, master/slave) with a high computation/communication ratio. Therefore, a primary concern for a broader use of Desktop Grids is the ability to address a wider scope of applications. This paper focuses on multi-parametric applications which feature a high volume of data inputs, highly shared among a large number of independent tasks. Such characteristics are frequent for trace-base simulations, bioinformatics applications which require access to large databases etc.

In this scenario of data-centric applications, the existing Desktop Grid Systems face a scalability issue. One should expect that more computing resources also provides more network bandwidth and storage capacity. On the contrary,

Desktop Grids Systems like BOINC or XtremWeb rely on a centralized data service architecture. For instance, data distribution with BOINC relies on multiple http servers and tasks are described as a list of files locations, which can be a potential bottleneck when scheduling tasks sharing large input files. To achieve high scalability, an efficient data distribution requires that the system can adapt: 1) to a very large number of resources, 2) to the high volatility of the resources, as computing nodes can join and leave the network at any time, 3) to the heterogeneity of network performances (nodes in a LAN, nodes at home) and 4) to the change of dimension of the system (daytime vs nighttime, flash-crowd effect).

Recent developments in content distribution such as collaborative file distribution (BitTorrent [8], Slurpie [9], Digital Fountain [10]) and P2P file sharing applications (EDonkey/Overnet [11], Kazaa [12]), has proven to be both effective, viable and *self-scalable*. Today, a significant part of the Internet bandwidth is used by P2P traffic [13]. The key idea, often featured as *parallel download* in the P2P applications, is to divide the file in a list of chunks. Immediately after a peer downloads a chunk from another peer, the peer serves the block for the other peers in the network, thus behaving itself as a server. Collaborative Content Distribution is a very active research topic and several promising strategies [14] such as the use of network coding [15], are proposed to improve the performance of the system. Real life observation [16] of a BitTorrent tracker during a five months period has shown that BitTorrent is immune to flash-crowd effect and was able to serve a 1.77GB file (the Linux RedHat 9 distribution) to 51000 users in less than 5 days.

However the context of Computational Desktop Grids shows specific characteristics when considering data involved in large multi-parametric applications. For instance, such applications are often composed of large number of small files describing the parameters. Thus efficiency of the data diffusion mechanism should be evaluated according to its impact on the overall performance of a parallel application when scheduled on the Grid.

To evaluate the potential of the collaborative data diffusion approach, we focus on the BitTorrent protocol. The two main arguments for the choice of BitTorrent are 1) it's a widely used cross-platform software and a *de facto* standard, 2) its P2P features are restricted to file transfer while other P2P softwares encompass indexing, naming and researching of files. Even

if this features might be appropriate in the context of data management, we consider that it is outside the scope of this paper. Thus, experimenting BitTorrent instead of other P2P protocols avoid to isolate the parallel download feature from the other features of file sharing application.

Our methodology for evaluating BitTorrent is the following: we first design a data management prototype using the XtremWeb Desktop Grid as a reference architecture. We discuss the feasibility of a BitTorrent-based Desktop Grid architecture with two qualitative criteria: security and deployment. Using this prototype we conduct 2 sets of performance evaluation. The first one aims at measuring the basic performance in terms of latency and bandwidth and evaluating the scalability of BitTorrent for file transfers. The second one tests BitTorrent against a synthetic bag of tasks parallel application with shared input data. We conduct those experiments within a 65-nodes cluster and compare the results with a classical FTP based solution used in centralized Desktop Grids. We demonstrate that even at that scale, BitTorrent improves the overall performance of the application by a factor up to 2.5.

The rest of the paper is organized as follows. Section 2 presents the challenge of data distribution for computational Desktop Grids. In section 3 we present our prototype. Then, in section 4, we conduct performance evaluations of BitTorrent. Finally we conclude the paper in Section 5.

II. COMPUTATIONAL DESKTOP GRID BASED ON COLLABORATIVE DATA DISTRIBUTION

To achieve high scalability, an efficient data distribution should address the following issues:

- 1) a very large number of resources. Internet PC Grids based on volunteers such as SETI@Home can reach several hundreds of thousands nodes. Thus, the system should avoid central data repositories which are potential bottlenecks.
- 2) the high volatility of the resources. Computing nodes can join and leave the network at any time. Associated with the large number of resources, it may preclude the computation of any fixed topology like broadcast trees for distributing the data.
- 3) the variation of the dimension of the system. The number and/or the location of nodes may vary according to patterns like day-time/night-time, work-days/weekend.
- 4) the heterogeneity of network performances between the nodes. Nodes may belong to the same LAN, or institution or be spread across Internet with intermittent connections. Typical ADSL users have asymmetric performance (uplink/downlink).
- 5) the *flash-crowd effect*. This phenomenon originally describes the sudden interest of the public for a file, which make the http server to collapse. The same kind of phenomenon is likely to happen for Computational Desktop Grids, when a new set of tasks is created and the data have to be spread to the computing nodes.
- 6) the size of the data. The system should provide both low latency for small messages and high bandwidth for bulk

data transfers.

- 7) self-organization. The system should organize itself an efficient diffusion on different topologies.
- 8) limited performance reduction in case of interferences.
- 9) fair share of Desktop Grid resources between concurrent transfers.
- 10) a limited resource consumption on the Desktop Grid nodes. The protocol to coordinate the file transfer should save the node bandwidth and the implementation should provide low CPU usage when communication overlaps computation [17].

A. An Overview of BitTorrent

BitTorrent is a popular file distribution system which aims at avoiding the bottleneck of FTP servers when delivering large and highly popular files. The key idea of BitTorrent is cooperation of the downloaders of the same file by uploading chunks of the file to each others.

A peer will first get all the informations about the file to download in a special static file called a `.torrent`. A `.torrent` contains the SHA1 signature for all the chunks composing the file and the location of a central agent called the *tracker*, which helps the peers to connect each other. In a BitTorrent network, trackers are responsible for keeping a list of informations about the peers: IP address, port to contact, file downloads. When a peer requests a file, it first asks the tracker a list of contact informations about the peers who are downloading the same file. The tracker does not organize transfers of the chunks between the peers; all data movements are decided locally by a peer according to local informations collected on its neighbors.

From this list of peers, the downloader asks its neighbors for the transfer of the file chunks. In the BitTorrent terminology, this operation of uploading is known as *unchoking*, the dual operation called *choking* is a temporary refusal of upload. Strategy for choking/unchoking relies on 3 rules: 1) no more than 4 file uploads are running at a time, 2) selection of the upload is based on the best transfer rates averaged on the last 20-second time frame and 3) once every 30 seconds, an *optimistic unchoking* operation is performed, which is a random selection of a fifth peers to upload. This operation helps to discover peers with a better bandwidth.

B. Related Works about BitTorrent

There have been several analysis of BitTorrent's performance through observation of large-scale utilization, through analytical modeling and through simulation of the protocol.

Measurement-based studies [18], [16], [8], [19] of BitTorrent are conducted by analyzing the logs of the tracker and instrumenting one client. Several trackers which correspond to different kind of files were investigated (the RedHat tracker for Linux distribution or SuprNova tracker for multimedia files). Thus data movements observed correspond to the interest users have in this files and are characterized in term of files availability and popularity, download volume and performance,

geographical analysis, evolution of the seeds/downloaders ratio. This observations conclude that BitTorrent is efficient to distribute large files when the number of nodes is high.

A model of BitTorrent based on fluid dynamic is proposed in [14], which quantifies the evolution of downloaders/seeders and the download time in function of nodes arrival/departure rates and network bandwidth. Authors conclude to the scalability of the BitTorrent and fairness of the built-in incentive mechanism.

Performance evaluation of BitTorrent through simulations, in [20], shows similar results and conclude also to efficiency under flash-crowd effect and fairness amongst peers in term of volume served.

In [15], Gkantsidis and P. Rodriguez propose to use network codings to improve performances of the BitTorrent protocol. With network coding, each node in the network is able to generate and transmit encoded pieces of data. Simulations shows an improvement of 2-3 times over unencoded version of BitTorrent.

Overall, our work contrasts to others in that we: (i) measure experimentally an instrumented BitTorrent protocol, (ii) use a small to medium scale deployment where the tradeoff between FTP and BitTorrent is likely to occur, (iii) use real traces of communication as input for simulations.

III. PROTOTYPE FOR A DATA DISTRIBUTION SUBSYSTEM

In this section we discuss the design of a prototype for a data distribution subsystem for Computational Desktop Grids. By designing this prototype, we want to evaluate the impact of BitTorrent on the Desktop Grid architecture.

While this subsystem aims at being integrated in the XtremWeb Desktop Grid, we think that it is generic enough to serve as a building block for new systems or as replacement for existing centralized data repositories like in BOINC. In this section we briefly introduce XtremWeb, then we present our prototype of data distribution subsystem and draw some guidelines concerning security, deployment and file management.

A. Overview of XtremWeb

XtremWeb [4] is an open source platform for Desktop Grids Computing. XtremWeb offers a software infrastructure to gather the unused resources of PCs (CPU, network, storage) spread over LANs or Internet to execute highly parallel applications. Its primary features permit multi-users, multi-applications and cross-domains deployments. XtremWeb follows the general vision of a Large Scale Distributed System turning a set of non specific resources (possibly volatile) into a runtime environment executing services (application modules, runtime modules or infrastructure modules) and providing volatility management. Typical parallel applications eligible to run on XtremWeb are embarrassingly parallel and an API is proposed to program them following the master/slave paradigm.

The architecture follows a three-tiers design (Worker, Coordinator, Client), which is commonly found in Desktop Grids

(see Condor [21] for a reference on this class of architecture). The design follows a set of three main principles: 1) a fault tolerance architecture allowing the mobility of the Clients, the volatility of the Workers and intermittent crashes of the Coordination service, 2) a set of security mechanisms based on autonomic decisions, 3) connection-less and one-sided communication protocol.

XtremWeb allows a set of Clients to submit task requests to the system which will execute them on Workers. The three-tiers architecture adds a middle tier between Client and Worker nodes. Thus there is no direct Peer-to-Peer task submission and file transfer between Clients and Workers. The role of the third tier, called the Coordinator, is a) to decouple Clients from Workers and b) to coordinate tasks execution on Workers. The Coordinator accepts task requests coming from several Clients, distributes the tasks to the Workers according to a scheduling policy, transfers application code and input file to Workers if necessary, supervises task execution on Workers, detects Worker crash/disconnection, re-launches crashed tasks on any other available Worker, collects and stores task results, delivers task results to Client upon request. In the present implementation, the Coordinator is implemented by a centralized node, eventually replicated. To ease the deployment phase regarding the connection issues raised by firewall and NAT configuration, all the communications are initiated by Clients and Workers toward the Coordinator node.

B. Architecture of the Data Distribution Subsystem

Our architecture follows closely the three-tiers architecture described above. We enhance the middle tier with two entities dedicated to data management: the Data Catalog and the Data Repository. Figure 1 illustrates this design.

Data Catalog keeps track of the datas and their location. Each data is referred by a unique identifier (UUID) computed with random number, time-stamp and network address, and additional attributes for managing the integrity of the data (SHA1 signature, size of the file, type flags: binary, executable, text, compressed, architecture dependant). Among this information, a data is associated to one or several Locators which are addresses of Data Repository and the needed informations to operate a file transfer (protocol, port, path, file name, login name and password).

Data Repository stores the data and should be remotely accessed by the senders and receivers of files. A Data repository can be managed by a Client of the Desktop Grid System to avoid transfer to an intermediate or it can be used as a reliable storage for managing fault tolerance of the Client (*launch and forget* mode). Data Repository run the necessary third party software required to access the data e.g. a FTP file server or a BitTorrent tracker.

C. Steps for a File Transfer

This section sketches the necessary steps to transfer a file from a Sender to a Receiver node.

Sender of a file first creates the data associated and computes its attributes to publish it on the Data Catalog. Next

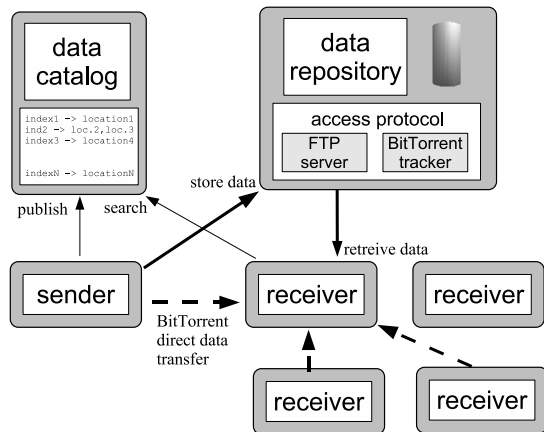


Fig. 1. Architecture of the prototype.

the data is uploaded from the Sender to the Data Repository. Obviously this operation depends on the protocol used for data transfer. For the FTP protocol, the Sender uses the regular sequence of commands: 1) log into the FTP server with the correct login and password, 2) move to the data directory, and 3) push the data to the server using the `put` command. The BitTorrent protocol imposes a different sequence. First the Sender creates the `.torrent` file using the file and the address of the tracker, and communicates it to the Data Repository. Both the Sender and the Data Repository simultaneously start a BitTorrent client. As the Sender gets the complete data, it is considered as a *seed* for the BitTorrent tracker, respectively, the Data Repository is considered as the *leecher*. When the operation is completed, the Data Catalog updates the index with a new location of this data.

Download of a file from the Data Repository by the Receiver follows the symmetric operations. The steps are equivalent for the FTP protocol, but `get` is issued instead of `put`. For the BitTorrent protocol, the Receiver will launch the BitTorrent client with the `.torrent` as parameter. But if others Receivers are downloading the same file, then the protocol ensures the concurrent transfers of the file chunks.

D. Some Remarks about Security, Deployment and Transfers Management

The design of this prototype raises some remarks about security, deployment and transfers management.

Data management subsystem offers a remote storage facility, thus it has to prevent abuse of the service by malicious users who could store their own files on the server. When a task has completed, informations to commit the results back to the Data Repository are available and could potentially be used to abuse the system. As a preventive action, the tasks submitter can retrieve the results as soon as a task is complete, therefore making the storage facility temporary and useless. Unfortunately, this solution might not be sufficient enough with FTP server as the right management allows to create and write new files in the server. On the contrary, BitTorrent doesn't allow push operation; transfers are only initiated by

download requests, which forbid unauthorized file creation.

Ease of deployment is a requirement for Desktop Grid Systems, it enforces that components are convenient to install and configure and that users can take advantage of existing infrastructure. Both BitTorrent and FTP implementations exist in multi-platform languages such as Java or Python which facilitates deployment on heterogeneous environment, and in native versions when performances are critical (server side). However, the security issues raised below by FTP server would suggest that the Desktop Grid System has a control over the right management of the server to create temporary user rights. This point could forbid location of the Data Repository on the Client side, and therefore direct Client-to-Worker communications.

If transfers are trivial to manage with FTP, BitTorrent imposes that nodes which own the data run BitTorrent to allow collaborative transfers. Thus, if a local mechanism exists to cache the data on the peer, the Data Repository should be able to trigger the BitTorrent client on the remote peer.

IV. EXPERIMENTAL RESULTS

This section presents the experimental evaluation of BitTorrent for Computational Desktop Grids. We begin with a description of the experimental conditions, then we present the results obtained in our experiments.

A. Experiments Setup

We have conducted our experiments in a predictable environment in order to evaluate the overhead and benefits of BitTorrent. The testbed is the *LRI Simulation Cluster* which is a 64-nodes cluster of heterogeneous ix86 machines. It's a set of single and dual CPU Athlon 32 1.5Ghz and Intel P-IV 2Ghz, each nodes with 885MB RAM and interconnected with a 100Mbps Ethernet switch. To stress the data server, it has been separated from the cluster. Due to the very dynamic nature of the BitTorrent protocol, every experiment was run 30 times, and the results present the averaged times.

The software configuration privileges Java implementation of the client part of the protocols in order to comply with realistic deployment on heterogeneous platform and the native implementation for the server part of the protocol. The BitTorrent implementation evaluated is Azureus version 2.2.0.2 (available on <http://azureus.sourceforge.net>), the BitTorrent tracker is the reference python implementation version 3.9.0 (available on <http://www.bittorrent.com>), the FTP client is provided by the Apache Jakarta commons-net package version 1.3.0 (available on <http://jakarta.apache.org>) and the FTP server is the Washington University FTP server version 2.6.2 (available on <http://www.wu-ftpd.org>).

B. Basic performance of BitTorrent

This first experiment compares the performance of BitTorrent versus FTP when distributing a file to a set of 20 nodes. The file size varies from 1 to 250 MB. Figure 2 presents the

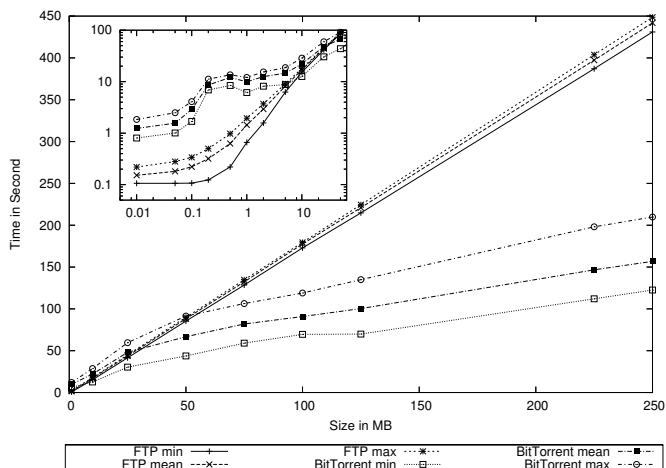


Fig. 2. The minimum, maximum and average completion time in second for the distribution of the file of a size varying from 1 to 250MB to 20 nodes. The figure presents a close-up of the latency at logarithmic scale for the distribution of small files, with a size comprised between 10KB and 50MB.

minimum, maximum and average completion time in seconds for the file transfer. The time is measured on each receiver node and is averaged over 30 experiments. The close-up figure plots with a logarithmic scale, the latency in second for transferring small files (size between 10KB to 50MB).

This first result illustrates that for large files, the time to complete the file distribution for FTP grows linearly with the size of the file. In this experiment the bandwidth of the FTP server is shared by all downloaders. The access list of the FTP server is configured to allow more downloaders than the actual number of available nodes. One can observe that the maximum, the average and minimum curves for FTP are very similar, which shows that the server bandwidth is equally shared between the downloaders.

BitTorrent clearly outperforms FTP when the file size is greater than 20MB. After this crossover point, the curve grows softly with a slope approximately equals to 0.45. The cooperation between the nodes is effective to decrease the transfer time even if a modest number of hosts is involved (in this case 20 nodes). This shows a clear potential of using BitTorrent for large file transfer instead of FTP. The difference between the minimum, mean and maximum curves is discussed later in the paper.

If BitTorrent protocol seems appealing for large file, FTP is more efficient for small files transfer. Multi-parametric applications are often composed of a simulation code associated with one or several configuration text files, which describe the parameters of the execution. Thus, this kind of studies implies the transfer of numerous small files.

When considering small file transfers, BitTorrent presents an overhead higher than FTP: respectively about 0.8 and 0.1 second. The BitTorrent overhead is due to the various steps the protocol imposes before actually starting the file communication. First the downloader has to read the `.torrent` file to extract the informations about the chunks and the tracker,

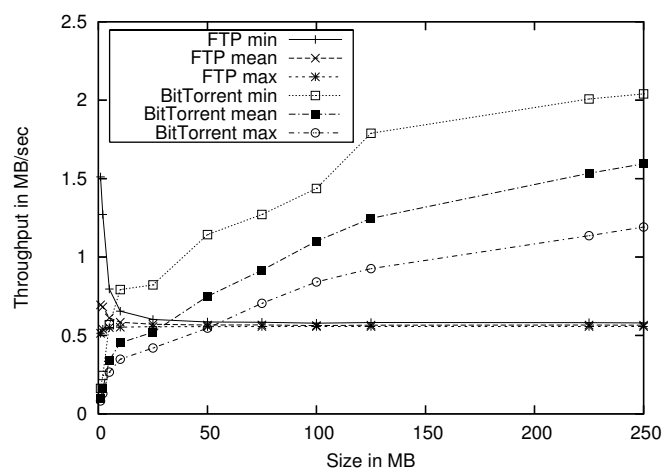


Fig. 3. Bandwidth (minimal, mean and maximal) in MB/s obtained when transferring a file to 20 nodes, with a size comprised between 1MB and 250MB.

next to contact the tracker to receive the list of peers. Finally the downloader needs to wait for the seeder or another peer to upload the chunks of file, with the additional constraint that upload queue is limited to 4 slots.

To cope with this overhead, Desktop Grids designers could: 1) use a dual protocol (FTP+BitTorrent) according to the size of the data, 2) or embody the small parameter file with the task description (this solution exists with XtremWeb).

Finally Figure 3 presents the bandwidth as measured locally on each node. We observe that the bandwidth for the FTP protocol is kept constant due to the fair sharing of the server bandwidth among the downloaders. In contrast, BitTorrent shows a noticeable improvement of the throughput when the file size increases. When the number of hosts is low, BitTorrent is effective for large file size.

C. Communication Patterns of BitTorrent

The following experiment compares the profiles of the file distribution of the two protocols. We plot in the Figure 4 the download times when a file of 100MB is delivered to 20 nodes. The experience is repeated 30 times and each point on the plot represents the time to completion for the file transfer for one node during one run. The horizontal axis represents the measures for every run. The upper three curves (maximum, mean and minimum) refer to the FTP measurements and the lower three ones to the BitTorrent measurements.

The figure shows that: 1) the download time is always lower for BitTorrent 2) while the distributions of the download time for FTP are quite homogeneous, BitTorrent suffers from less consistent performance. With BitTorrent, the download time can vary from a factor 3 between the fastest and the slowest node.

In order to understand this variations we have instrumented the BitTorrent client. On each peer, we log every communications made to the other peers. The Figure 5 presents the communication pattern of the BitTorrent protocol when

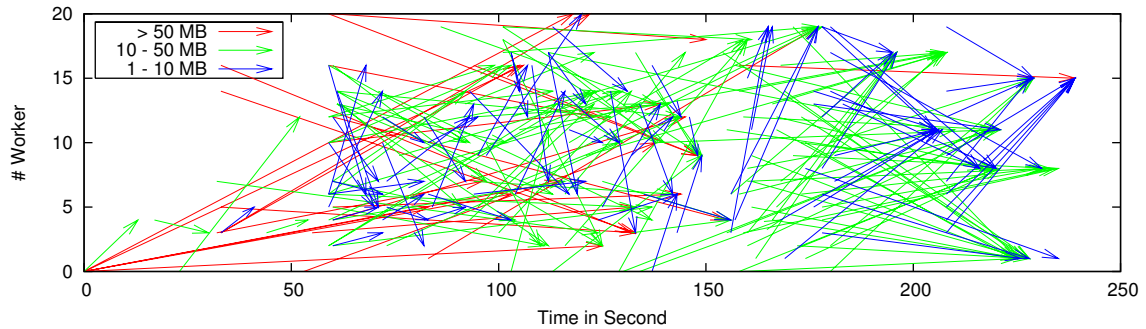


Fig. 5. Communication pattern of the BitTorrent protocol when diffusing a 250 MB file to 20 nodes. First point of a vector presents the beginning of a communication (start time, sender) and the second point presents the end of the communication. The color of the vector presents the volume of data transmitted.

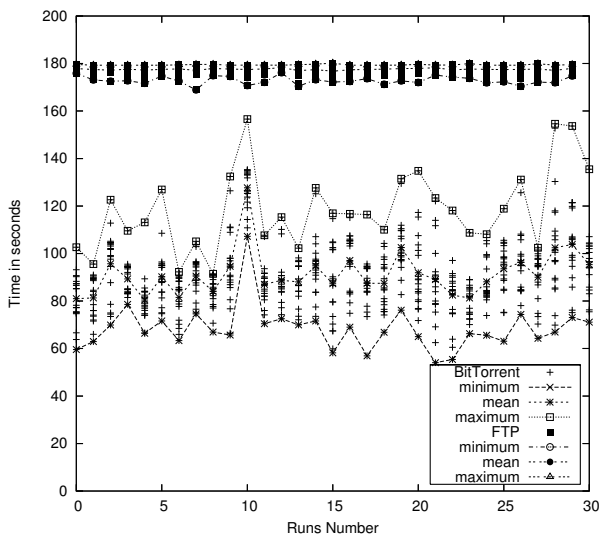


Fig. 4. Profiles of downloads for a 100 MB file by 20 machines: each point on the plot represents the time to completion for the file transfer for one node during one run. The three curves represent the maximum, mean and minimum for both FTP and BitTorrent.

diffusing a file of size 250 MB to 20 nodes. A vector represents a communication from a sender to a receiver (vertical axis) during a period of time (horizontal axis) with no more than 10 second idle. We discarded vectors with a volume of data is less than 1MB, however more than 99.45% of the total volume transmitted is presented.

The figure shows 1) nodes start to upload file chunks to other nodes before receiving the whole file, 2) largest communications are performed at the start of the diffusion and 3) the topology at the beginning of the diffusion represents a tree and a pipeline is organized to transmit the whole file to the last served nodes.

Future works should try to anticipate the construction of the topology in order to model the downloading time of individual nodes. This point can impact the overall performance of application execution, as forecasting of a communication duration is required for appropriate scheduling decisions.

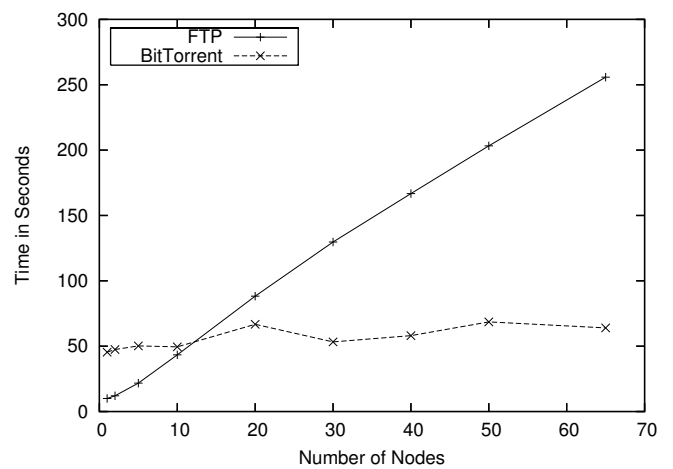


Fig. 6. Completion Time for the distribution of a 50MB file to a varying number of nodes. The vertical axis shows the time in second, the horizontal axis shows the number of nodes.

D. Scalability Evaluation

With this experiment, we evaluate the scalability of BitTorrent compared to FTP. The benchmark consists of a pool of nodes varying from 1 up to 64 which simultaneously download a 50MB file. The file is located on a central FTP server or on the first BitTorrent client. Each node measures the time to complete the download and the Figure 6 presents the average time to transfer the file, compared to the number of workers.

As seen in the figure the download time of BitTorrent remains stable as the number of resources increases while it linearly increases with FTP. This results shows that the scalability of BitTorrent is the one expected when the number of nodes is high. Other studies based on simulations implying up to 5000 nodes confirm this general trend. However, with a 50 MB file, there is a crossover point around 10 workers where FTP is more efficient than BitTorrent due to the overhead of BitTorrent.

The previous experiments were considering a set of workers starting the file transfer at the same time. This scenario is very unlikely to happen in the real life. Systems like XtremWeb and BOINC try to avoid the situation where all the participants

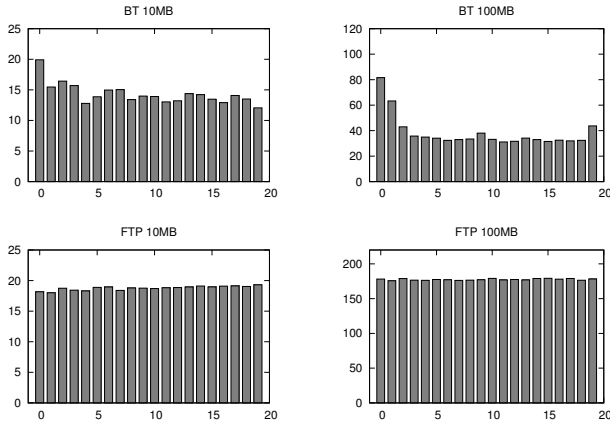


Fig. 7. Non-simultaneous downloads.

simultaneously connect to the server and make it collapse. The solution most commonly used (by XtremWeb and BOINC) is that the client to server communications use an exponential back-off in the case of failure of the server.

The following experiment considers the distribution of two files (10 and 100MB) to a set of 20 nodes. The workers start their download each one after the other, with a waiting time of 1 minute. The figure 7 shows the downloading time measured by each nodes and sorted by the starting time of the download. The leftmost bar on the graph represents the first host to begin the data transfer, and the rightmost bar the last node, which will start the transfer 20 minutes after the first one.

As seen on the figure, the transfer time for a file of 10MB is below 1 minute (between 12 and 20 seconds according to the protocol), so every transfer are completed before the next one start. The figure shows that the time to complete the first transfer of the file is a little higher for BitTorrent (20s) than FTP (18s) but as more and more copies of the file are distributed to other nodes, the download time for BitTorrent decreases by a factor 1.9 when the download time for FTP stays the same.

When considering a larger file (100MB) the transfer time can be more than 1 minute. An interesting feature of BitTorrent is that the time for the first download is also decreased compared to FTP (respectively 80 versus 170 seconds) which demonstrates the efficiency of the BitTorrent block allocations strategy.

E. Evaluation on a synthetic multi-parametric application

The last set of experiments evaluate the potential of using BitTorrent in place of an FTP server for a multi-parametric synthetic application.

First, we consider an application with several computing/communication ratios. The application is composed of a set of n independent tasks, n being equal to the number of involved nodes. A task consists of two phases: a file transfer (20MB) followed by an execution. Execution time of each task is $t_{communication} + t_{computation}$. The reference $t_{communication}$ is set to the time to transfer 20M between 2 nodes using FTP

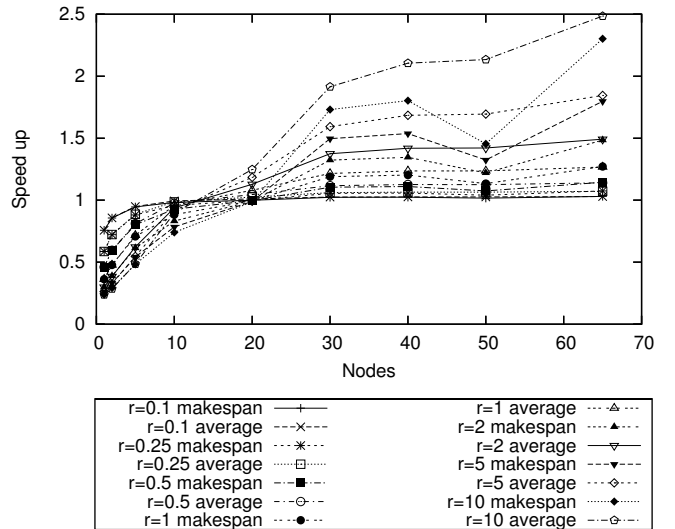


Fig. 8. Speed up of BitTorrent versus FTP on a synthetic application. We plot curves for ratios communication/computation r equal to 0.1, 0.25, 0.5, 1, 2, 5, 10 and two metrics for the execution time *makespan* and *average*.

and the ratio $t_{computation}/t_{communication}$ varies from 0.1 to 10. We have used two metrics for computing the speed up: *makespan* is the duration between the start of the first task and the completion of the last task. Nodes could be idle when the last nodes are still computing and thus be attributed to an other application. We have used a second metric *average* which represents the average execution time on each node.

The result shown in Figure 8 presents the speedup of BitTorrent compared to FTP for a varying number of nodes. Speed-up of BitTorrent increase with the number of nodes and the communication ratio to reach a factor 2.5 when r is 10 and n is 70 with *average* metric and 2.3 when considering *makespan*. We note that *average* shows better results than *makespan* which is due to the differences in downloading time between the nodes (*makespan* considers the maximum downloading time). On the contrary, when the number of nodes is small and when the communication ratio is high, FTP outperforms BitTorrent due to the large overhead when transmitting small files.

In the second application, we want to evaluate the effect of file sharing amongst tasks distributed by BitTorrent. We consider a set of files equals to the number of nodes, and d the difference ratio between the files. Table 9 shows that BitTorrent's performance are improved by a factor 1.2 and 2.2 when distributing a unique file ($d = 0.05\%$) compared to a set of files, all different ($d = 100\%$).

difference ratio	0.05 %	25 %	50 %	75 %	100 %
5 MB	9.94	17.21	19.9	20.71	21.92
25 MB	66.46	79.41	84.68	84.88	86.37

Fig. 9. Effect of file sharing amongst tasks on BitTorrent distribution time to distribute a set of files to 20 nodes according to the difference ratio.

The last experiment confronts BitTorrent and FTP against

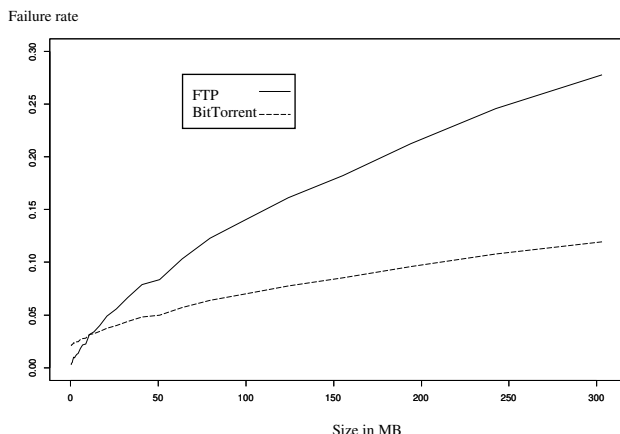


Fig. 10. Transfer failure rate versus transfer size (in MB)

node volatility. This experiment evaluates the transfer failure rate, that is the probability that a host will become unavailable before a transfer complete. We base this model on host availability characterization proposed by Kondo and All. [22] and a linear model of the results presented in Figure 2. The higher efficiency of BitTorrent permits to decrease the transfer failure rate by a factor 2.8.

V. CONCLUSION

Collaborative Data Distribution has become a key technology in the Internet. In this paper, we investigated BitTorrent as a protocol for Data Diffusion in the context of Computational Desktop Grid. We designed a prototype and found that even if Desktop Grid architectures often rely on centralized coordination components, they can easily integrate this P2P technology without fundamental changes on their model of security or deployment.

We conduct experimental performance evaluations of the protocol on a LAN cluster and showed that BitTorrent is efficient for large file transfers, scalable when the number of nodes increases but suffers from a high overhead when transmitting small files. Comparison with FTP-based centralized Desktop Grid on the execution of a multi-parametric application demonstrates that BitTorrent is able to execute application with a higher communication/computation ratio and to reduce the fault probability, which are two requirements for a broader use of Desktop Grid. However, due to its high overhead, a misuse of BitTorrent, e.g. for small files or files that are not shared enough, lead to a sensible performance penalty.

Therefore we think that future works around integration of Desktop Grid and Collaborative Data Distribution should focus on: 1) improve the latency of BitTorrent, 2) experiment with other P2P protocols and evaluate the cost and benefit of indexing/publishing/searching data, 2) design multi-protocols Data Desktop Grid Systems, 3) experiment with other deployments (ADSL/Multi-LAN/WAN) and investigate how BitTorrent performs with various physical topologies and 4) design BitTorrent-aware scheduling strategies for data-centric applications.

REFERENCES

- [1] D. Anderson, S. Bowyer, J. Cobb, D. Gedye, W. T. Sullivan, and D. Werthimer, "A New Major SETI Project Based on Project Serendip Data and 100,000 Personal Computers," in *Astronomical and Biochemical Origins and the Search for Life in the Universe, Proc. of the Fifth Intl. Conf. on Bioastronomy*, 1997.
- [2] Distributed.net, "RSA Labs' 64bit RC5 Encryption Challenge," <http://www.distributed.net>.
- [3] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, November 2004.
- [4] F. Cappello, S. Djilali, G. Fedak, F. M. Thomas Hérault, V. Néri, and O. Lodygensky, "Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid," *FGCS Future Generation Computer Science*, 2004.
- [5] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing," in *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003.
- [6] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropy: Architecture and Performance of an Enterprise Desktop Grid System," *Journal of Parallel Distributed Computing*, vol. 63, no. 5, pp. 597–610, 2003.
- [7] W. Gentszsch, *Sun Grid Engine (SGE): A cluster resource manager*, 2002.
- [8] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.
- [9] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," in *Proceedings of IEEE INFOCOM*, March 2004.
- [10] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital-Fountain Approach to Reliable Distribution of Bulk Data," in *proc. of the ACL SIGCOMM*, 1998.
- [11] EDonkey, "Edonkey, overnet homepage," January 2002, <http://www.edonkey200.com/>.
- [12] FastTrack, "P2P Technology. KaZaA Media Desktop," January 2002, <http://www.fasttrack.nu/>.
- [13] CAIDA, "CAIDA, the Cooperative Association for Internet Data Analysis: Top applications (bytes) for subinterface 0[0]: SD-NAP traffic," 2002.
- [14] D. Qiu and R. Srikant, "Modeling and Performance analysis of BitTorrent-like Peer-to-Peer Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 367–378, 2004.
- [15] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proceedings of IEEE/INFOCOM 2005*, Miami, USA, March 2005.
- [16] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime," in *Proceedings of Passive and Active Measurements (PAM)*, 2004.
- [17] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrante, "On the interference of communication on computation in java," in *the 3rd International Workshop on Performance Modeling, Evaluation and Optimization on Parallel and Distributed Systems (PMEO-PDS'04)*, Santa Fe, New Mexico, April 2004.
- [18] J. A. Pouwelse, P. Garbacki, D. Epema, and H.J.Sips, "A Measurement Study of the BitTorrent Peer-to-Peer File Sharing System," Delft University of Technology, Tech. Rep. PDS-2004-007, 2004.
- [19] B. N. L. Anthony Bellissimo, Prashant Shenoy, "Exploring the Use of BitTorrent as the Basis for a Large Trace Repository," University of Massachusetts, Tech. Rep., 2004.
- [20] A. R. Bharambe, H. Cormac, and V. N. Padmanabhan, "Understanding and Deconstructing BitTorrent Performance," Microsoft Research, Tech. Rep., 2005.
- [21] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*. Washington, DC: IEEE Computer Society, 1988, pp. 104–111.
- [22] D. Kondo, M. Tauber, C. L. Brooks, H. Casanova, and A. Chien, "Characterizing and evaluating desktop grids: An empirical study," in *IPDPS 2004, IEEE/ACM International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico, 2004.