

Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent

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 software for collaborative content distribution over Internet offers a new opportunity for efficient and fast delivery of high volume of data. In a previous paper, we have investigated BitTorrent as a protocol for data diffusion in the context of Computational Desktop Grid. We 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. This paper investigates two approach to overcome these limitations. First, we propose a performance model to select the best of FTP and BitTorrent protocols according to the size of the file to distribute and the number of receiver nodes. Next we propose enhancement of the BitTorrent protocol which provides more predictable communication patterns. We design a model for communication performance and evaluate BitTorrent-aware versions BT-MinMin, BT-MaxMin and BT-Sufferage scheduling heuristics against a synthetic parameter-sweep application.

I. INTRODUCTION

Computational Grids [1] and Desktop Grids [2], [3], [4], [5], aggregate resources (CPU, network, storage) to execute large and highly parallel applications. These systems have been proven efficient for CPU-intensive applications. For instance SETI@Home [6] successfully gathers tens of thousands nodes and claims a tremendous 100 TFlops computing power as of January 2005. One should expect that more computing resources also provide more network bandwidth and storage capacity. On the contrary, systems like BOINC or XtremWeb rely on a centralized data service architecture which can be a potential bottleneck in a large, heterogeneous and dynamic environment.

Recent developments in content distribution such as collaborative file distribution (BitTorrent [7], Slurpie [8], Digital Fountain [9]) and P2P file sharing applications (EDonkey/Overnet [10], Kazaa [11]), has proven to be both effective, viable and *self-scalable*. 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 [12] such as network coding [13], are proposed to improve the performance of the system. Real life observation [14] 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.

Our previous work [15] investigated BitTorrent as a protocol for Data Diffusion in the context of Computational Desktop Grid. Experimental performance evaluations of the protocol on a LAN cluster 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 helps executing applications with a higher communication/computation ratio. 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 significant performance penalty.

There exists several heuristics to distribute independent tasks sharing large files onto the Grid [16], [17], [18], [19], [20]. However, the distributed nature of collaborative file distribution make the data transfer less predictable. This paper investigates BitTorrent-aware scheduling heuristics for distributing independents tasks sharing large data. First we augment our previous experimental measures of the BitTorrent protocol to obtain a full medium scale comparison of BitTorrent versus FTP. Based on these observations we propose an analytical model of the performance which helps to select the best of the two protocol according to the size of the data and the number of receiver nodes. Next, we modify some aspect of the BitTorrent protocol and we propose enhancements of existing heuristics to schedule parameter sweep applications with dual-protocol communications, predictive communications ordering. We demonstrate that our scheduling improves the overall performance of the application by a factor up to 3 compared with a basic Round Robin heuristic with FTP.

This paper is organized as follows. In Section 2 we present our system model for application and data diffusion. Section 3 presents the BitTorrent protocol. Based on these observations we propose heuristics for tasks scheduling in Section 4 and evaluate their performance in Section 5. In the conclusion, we summarize what we have learned in this study.

II. SYSTEM MODEL

This section describes the distributed system investigated and introduces terminology and performance metrics used to conduct performance evaluation.

A. System Environment

Existing computational grid systems like BOINC, Entropia, XtremWeb rely on a centralized architecture for distributing both tasks and data. To avoid this potential bottleneck, we propose to modify the architecture by introducing a collaborative data diffusion approach where all participating nodes cooperate to data transfer. Note that the scheduling of the tasks is still managed in a centralized way. Figure 1 illustrates the difference with the fully centralized approach versus the centralized tasks scheduling combined with collaborative data diffusion approach.

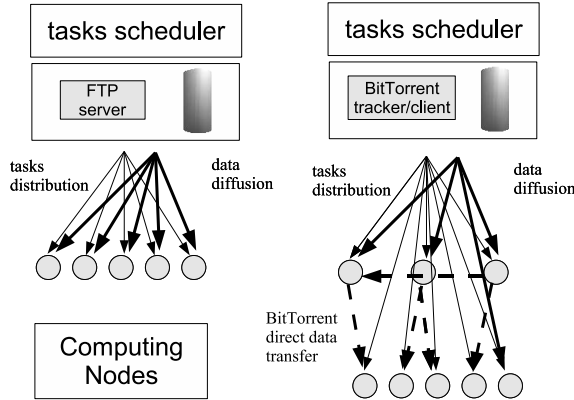


Fig. 1. Comparison of two architectures for scheduling tasks and data: centralized tasks and data scheduling versus centralized tasks scheduling and collaborative data diffusion.

To evaluate potential of collaborative data-diffusion, we focus our study on parameter-sweep applications sharing large input data. Efficient scheduling heuristics should adapt to the following constraints:

- data heterogeneity. Multi-parametric applications are often composed of large number of tasks, each one uniquely described by a small parameter text file. On the other extreme, a unique large input file can be shared by every task.
- different characteristics and performance of file transfer protocol. Collaborative protocol are scalable with a large number of nodes or large data, but the distributed nature of the protocol introduces a high latency overhead. Scheduling heuristics should select the best protocol according to the volume of data to distribute and the number of receiver nodes.
- unpredictability of data transfer. Transfers organized locally by the peers, make prediction of transfer time more difficult and inaccurate for a central scheduler. This can impact the quality of scheduling heuristics involving communication times in their evaluation.

B. 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 other.

A peer will first get all the information 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 information 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 information 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 information 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.

C. Scheduling and Performance Metrics

We consider the execution of a data-intensive parameter-sweep application. An execution of the application is the executions of a collection of tasks on a set of processors. We consider an application \mathcal{A} composed of a non empty set of independent tasks:

$$\mathcal{A} = \{T_1, \dots, T_n\}, \text{ and } n > 0 \quad (1)$$

Each task is characterized by a number of operations Ω_i . Furthermore, to each task T_i is associated one input dataset $\Theta(T_i) = \{I_1^i, \dots, I_l^i\}$ where each I_j^i is an input data. The result of the execution of a task is one output data O_i .

Thus, the global application input data set is:

$$\Theta(\mathcal{A}) = \bigcup_{1 < i < |T|} \Theta(T_i) \quad (2)$$

We define the Grid \mathcal{G} as a set of k heterogeneous processors P connected by a homogeneous network.

$$\mathcal{G} = \{P_1, \dots, P_k\}, \text{ and } k > 0 \quad (3)$$

A processor has a computing power capacity expressed by the number of operations per seconds Ψ_k processed.

We consider four parameters to characterize the application:

- The first parameter, *Application Grain (AG)* is specified by the pair: size of input data and number of processors in the Grid.

- The second parameter is the *Application Size (AS)* which defines the number of tasks of the application relative to the number of available processors.
- The next parameter characterizes the *Computation/file IO Ratio (CIOR)*. To be protocol agnostic when determining this ratio we have translated into a ratio size of input data/sum of number of operations.

$$CIOR = \frac{\text{size}(\Theta(\mathcal{A}))}{\sum_{1 < i < |T|} \Omega_i} \quad (4)$$

- The last parameter, *Shared Data Ratio (SDR)* quantifies the amount of data which are shared amongst the tasks. *SDR* is computed as the volume of application input data over the volume of input data distributed to each task:

$$SDR = \frac{\text{size}(\Theta(\mathcal{A}))}{\sum_{\substack{1 < i < |T| \\ 1 < k < |I(T_i)|}} \text{size}(I_i^k)} \quad (5)$$

The rationale with this parameter is that collaborative transfer should be more efficient when data are highly shared. The lower *SDR* is, the more data are shared; a *SDR* of 1 meaning that no files are shared between tasks.

The schedule Σ_A of application \mathcal{A} is the schedule of every task T_i to a processor. The expected execution time of a task T_i on a processor P_j is defined as the time taken by P_j to execute T_i given the number of operation Ω_i and the computing power Ψ_i . The expected transfer time of the data input set $\{I_1^i, \dots, I_k^i\}$ is the time to deliver the input data of task T_i locally to the processor P_j before the task can start its execution. We don't assume any remote storage nor remote caching facilities, but we assume an infinite local cache on every processor. Finally the expected completion time c_i for a task T_i on processor P_j is defined as the wall-clock time at which T_i completes, given that every data are present and all previously assigned task are finished.

We measure the application execution time using the *makespan* [21] metric. The *makespan* of the complete schedule is the maximum of all expected completion times $\max(c_i)$. Intuitively the *makespan* is the time elapsed between the first task is assigned and the last task is finished. We compare the scheduling heuristics by discussing their ability to minimize the *makespan* metric.

D. Related Works

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

Measurement-based studies [22], [14], [7], [23] of BitTorrent are conducted by analyzing the logs of the tracker and instrumenting one client. Data movements observed are characterized in term of files availability and popularity, download volume and performance, geographical analysis, evolution of the seeds/downloaders ratio. These 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 [12], which quantifies the evolution of downloaders/seeder and the download time in function of nodes arrival/departure rates and network bandwidth. Authors conclude to the scalability of the BitTorrent protocol and fairness of the built-in incentive mechanism. Performance evaluations of BitTorrent through simulations [24], [13] show similar results and conclude also to efficiency under flash-crowd effect and fairness amongst peers in term of volume served.

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 an input for a Grid simulator.

A large number of research paper address the issue of scheduling independent tasks sharing files onto an homogeneous or heterogeneous set of computing nodes. Relative to data movement, the first simplifying assumption made is that the scheduling heuristics have a full knowledge of data movements and static information about network characteristics [17]. In highly dynamical environment, such as Grid infrastructure, network performance is likely to vary in time. Thus heuristics have been proposed with dynamic network information provided by Grid network monitoring software such as NWS [25]. Xsufferage [19] is an enhancement of heuristics proposed in [26] to schedule parameter sweep applications with file I/O requirements. Storage Affinity [20] extends these heuristics with accurate data placement and replication heuristics. AdRM [16] is a regression model to predict transfer time of remote data files.

The recent development of collaborative data distribution adds specific constraints when designing scheduling heuristics for data intensive application. The communication pattern of BitTorrent is still poorly understood, and there is no analytical model of the protocol which provides forecasting of transfer time for every receiver node. Our work enhances the BitTorrent built-in incentive mechanism in order to provide predictive communication sequencing, which is the base for designing BitTorrent-aware scheduling heuristics.

III. BITTORRENT ANALYSIS

This section presents an experimental analysis of BitTorrent when distributing large data. 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 compared to FTP protocol. 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 node 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-ftp.org>).

B. Experimental Analysis

This first experiment compares the performance of BitTorrent versus FTP when distributing a file to a set of nodes whose size ranges from 5 to 60. The file size varies from 1 to 250 MB. Figure 2 presents the average completion time in seconds for the file transfer. The time is measured on each receiver node and is averaged over 30 experiments.

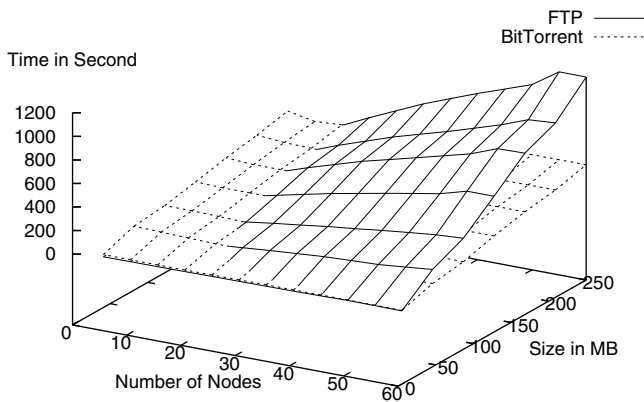


Fig. 2. Performance comparison of FTP and BitTorrent. The two curves present the average completion time in seconds for the distribution of a file of a size varying from 5 to 250MB to a set of nodes whose size varies from 5 to 60.

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.

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.

As seen in the figure the download time of BitTorrent remains stable as the number of resources increases while it linearly increases with FTP. These results shows that the scalability of BitTorrent is the one expected when the number of nodes is high. Other studies based on simulations including 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.

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.

Figure 3 presents 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 close-up figure plots with a logarithmic scale, the latency in second for transferring small files (size between 10KB to 50MB).

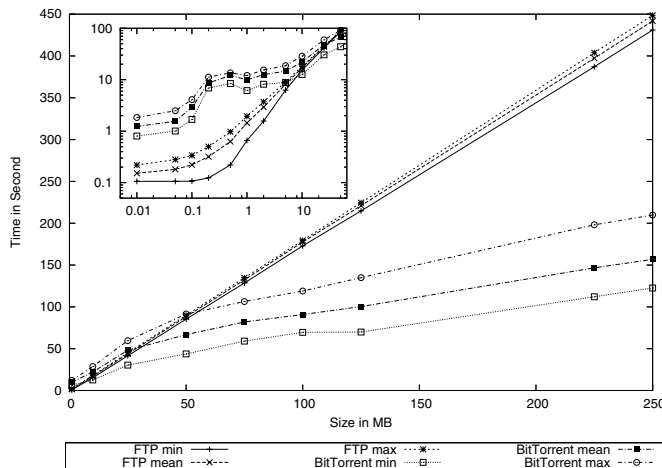


Fig. 3. The minimum, maximum and average completion time in second for the distribution of the file of a size varying from 1 to 250MB to 20 nodes. The figure presents a close-up of the latency at logarithmic scale for the distribution of small files, with a size comprised between 10K and 50MB.

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 information about the chunks and the tracker, 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.

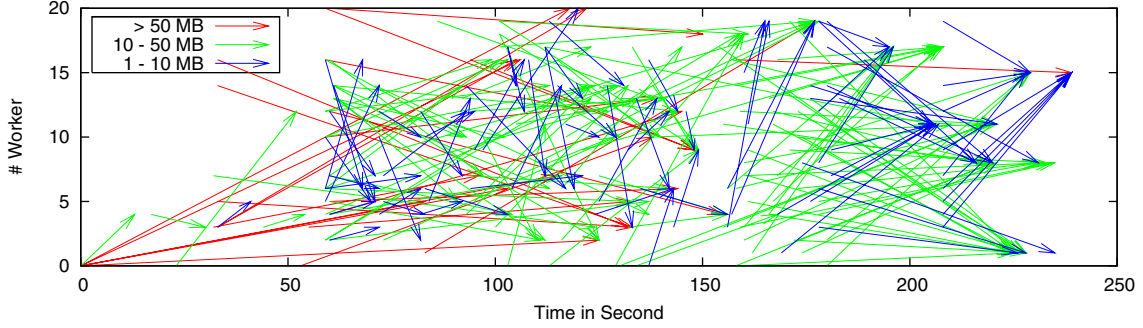


Fig. 4. Communication pattern of the BitTorrent protocol when diffusing a 250 MB file to 20 nodes. First point of a vector presents the beginning of a communication (start time, sender) and the second point presents the end of the communication. The color of the vector presents the volume of data transmitted.

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).

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. 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.

C. Communication Patterns of BitTorrent

In order to understand these variations we have instrumented the BitTorrent client. On each peer, we log every communication made to the other peers. The Figure 4 presents the communication pattern of the BitTorrent protocol when 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 seconds 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 that: 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.

D. Choosing the Best of the Two Protocols

The analytical performance analysis of BitTorrent and FTP aims at determining the cross-over points, where BitTorrent should be use instead of FTP. We want to express these points in terms of number of nodes and file size.

With the FTP protocol, the bandwidth is considered equally shared amongst the downloader. Thus the time to achieve a file transfer with FTP by a node is:

$$t(N, S) = \alpha_{ftp} + \frac{N.S}{\beta_{sender}} \quad (6)$$

where α_{ftp} and β_{sender} are two constants respectively the latency of the FTP protocol and the bandwidth capacity at the sender node; S is the size of the data and N the number of receiver nodes.

The empirical observation of the BitTorrent communication patterns lead us to modelize the distribution time as a tree:

$$t(N, S) = (\alpha_{bittorrent} + \frac{a.S}{\beta_{sender}}).log_a(N) \quad (7)$$

where $\alpha_{bittorrent}$, β_{sender} and a are three constants respectively the latency of the BitTorrent protocol, the available bandwidth by the sender node and the a the number of leaves of the tree as there are 5 upload slots; S is the size of the data and N the number of receiver nodes.

The next figure presents the evaluation of the model against experimental values. When the prediction of model is wrong i.e. when model inaccurately select the minimum of the completion time provided by the two protocol, we plot the communication overhead. The overhead is presented as a ratio compared to the best solution. As seen on Figure 5, the heuristic performs 92.8% correct predictions. When the prediction is wrong, the communication overhead ranges from 2.2% to 18.3%.

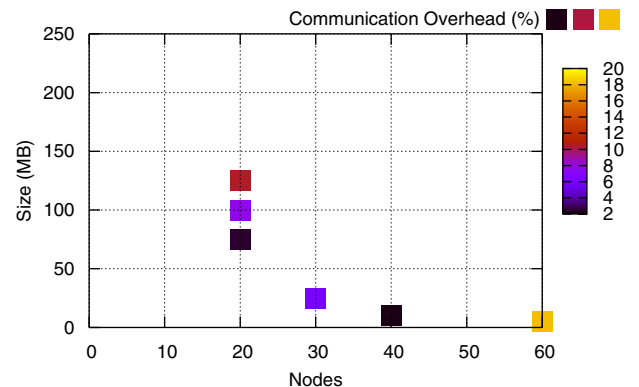


Fig. 5. Evaluation of the performance model to select the best of the two protocols against experimental values.

E. Predictive Communications Ordering

	Min.	1st Qu.	Mean	3rd Qu.	Max.
small	30.82	34.19	36.97	39.01	45.39
sm. PCO	13.64	19.98	25.23	29.10	37.55
medium	93.37	101.50	105.00	107.40	115.60
med. PCO	41.18	56.27	64.81	75.16	89.59
large	118.5	139.9	145.0	152.3	157.8
lar. PCO	61.19	78.32	89.36	101.20	122.20

Fig. 6. Evaluation of predictive communication ordering for 3 file distributions: small (25MB, 20 nodes), medium (75MB, 40 nodes) and large(250MB, 50 nodes). The figure reports the minimum, average, maximum, 1st and 3rd quartile of distance between the predetermined rank and the actual observed rank. Lines with PCO reports values for BitTorrent with predictive communication ordering.

Efficient scheduling heuristics needs to evaluate data transfer times. The distributed nature of BitTorrent makes this prediction difficult as each host locally decide the communication based on the state of their neighbors. We introduce a modification of the built-in incentive mechanism to order the communication pattern in a more predictable fashion.

With the BitTorrent vanilla policy, every host requests chunks of file from list of downloaders, but one host can only send chunks to 5 downloaders at a time. The policy favors hosts which can achieve the best upload rate and exchange first the rarest chunks of a file. We modify the best downloader selection policy to select hosts according to a fixed and predetermined criteria.

The next experiment evaluate the efficiency of predictive communication ordering. We run 30 experiments with and without communication ordering and report for every node the time to complete the file transfer. Hosts are ranked according to the file transfer completion time. The *distance* between the predetermined rank and the rank achieved by the experiment evaluates the efficiency of predictive communication ordering. Figure 6 summarizes the results by comparing the distances achieved without and with predictive communication ordering (PCO) for 3 files distributions schemes. Figure 6 shows that PCO is able to lower the distance, which means that nodes finish their download closer to their expected rank.

IV. SCHEDULING STRATEGIES INVESTIGATED

We investigate two levels of scheduling heuristics: *basic heuristics* for reference and *knowledge-based heuristics* which require knowledge about communication performance and CPU load performance

A. Basic Scheduling Heuristics

The reference scheduling strategies implemented is the **Round Robin** heuristic. This evaluation aims at: (i) measuring the potential of BitTorrent against FTP, and (ii) serving as a reference to compare advanced heuristics.

We have implemented the Round Robin in three flavors: **FTP-RR** statically assigns tasks to hosts and realizes the file transfers using the FTP protocol, **BT-RR** is the same heuristic

but file transfers are managed with BitTorrent, **DP-RR** is the dual protocol version of the same heuristic where the protocol is selected according to the analytical mode presented above.

B. Knowledge-based Scheduling Heuristics

Min-Min, *Max-Min*, and *Sufferage* are three scheduling heuristics widely studied in the context of Grid, which rely both on CPU and network performance knowledge. These three heuristics works as follow: a) for every task to be scheduled, a host is assigned which minimizes the expected completion time, b) pick one couple task and host according to a specific *metric* and iterate until all tasks have been scheduled. The process repeatedly build a schedule plan which drives the execution of the application onto the Grid. The heuristics differs by the *metric* function used to select the task to schedule: *Min-Min* is the minimal completion time, **Max-Min** is the maximal completion time, and **Sufferage** is the maximal difference between the earliest and the second earliest completion time. More details and discussions about these heuristics can be found in [26], [19], [20].

We propose BitTorrent aware variants (**BT-MinMin**, **BT-MaxMin**, and **BT-Sufferage**) based on predictive communications ordering. As these three heuristics rely on forecasting of communication times for every receiver nodes, we generate a sequence of file transfer times. Nodes are assumed to finish their transfer at regular interval between the minimum and maximum observed times.

V. PERFORMANCE EVALUATION

In this section, we analyze the performance of BitTorrent-aware scheduling heuristics. We have used simulations to evaluate the performance of the scheduling algorithms. Since, the performance of the scheduler is strongly dependant of the characteristics of the application, we have issued numerous simulations with varying application grain, size, CIOR and SDR parameters.

A. Methodology

To reach the highest degree of reality, the simulator is based on real network and cpu traces obtained on a cluster. To simulate the network, we use the file transfer completion times presented in section II. For each distribution of a file, the simulator uses the transfer completion times for every receiver node involved in the distribution. Each simulation uses 30 measures of different file distributions. The CPU performance of each node is modeled by running the CPU availability measurement tool presented in [27], collected during a period of 51 days. From these traces, we observed two distinct behaviors upon another process is run on the nodes. For simulations, CPU load samples are randomly picked-up among a set of traces after discarding traces when the host is unavailable.

To determine the computation/communication ratio of the application, we calibrate CIOR by fixing the following equality: time to transfer one file with FTP between two hosts equals the execution time by the fastest host. From this equality,

we deduce a reference CIOR expressed in bytes/operations. Tasks are defined by a number of operations to execute, which follows a normal distribution modeled by mean determined according to the desired CIOR factor, and variance equals to a tenth of the mean. For every simulation we run 27000 experiments and present the averaged results.

B. Comparison of Scheduling Strategies

1) *Application Granularity*: First we compare the scheduling according to three application grains ; *small* (25MB, 20 nodes), *medium* (75 MB, 40 nodes) and *large* (250MB, 50 nodes). We fix the CIOR factor to 65.1 and the data input set consists of a unique file shared amongst the tasks. The size of application is set to 1, that is the number of tasks equals the number of processors in order to stress the data diffusion phase of the application execution.

In Figure 7 we can see that smaller granularities are not significantly impacted by the protocol. For large grain, FTP-based heuristics clearly under-performs due to the high communication overhead of the FTP protocol. For larger granularities, we further observe that knowledge-based scheduling heuristics outperforms round-robin ones.

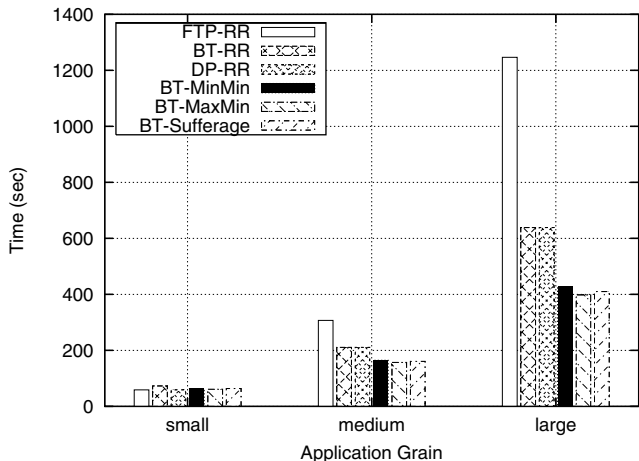


Fig. 7. Evaluation of scheduling heuristics according to Application Granularity.

2) *Application Size*: Figure 8 compares the scheduling heuristics against a varying application size. The application size is the number of tasks relative to the number of processors. In these experiments we schedule 40 to 400 tasks to 40 processors, every task share a unique input file of 75MB.

We first observe that knowledge-based heuristics always perform better than basic heuristics. More, the slope of knowledge-based performance curves is softer when application size increases.

On the bottom leftmost part of the figure, we observe a plateau on the execution time for knowledge-based scheduling heuristics. This is due to the high variation of communication time: the latest node which receives its data will not execute more tasks when the application size vary from 1 to 2.

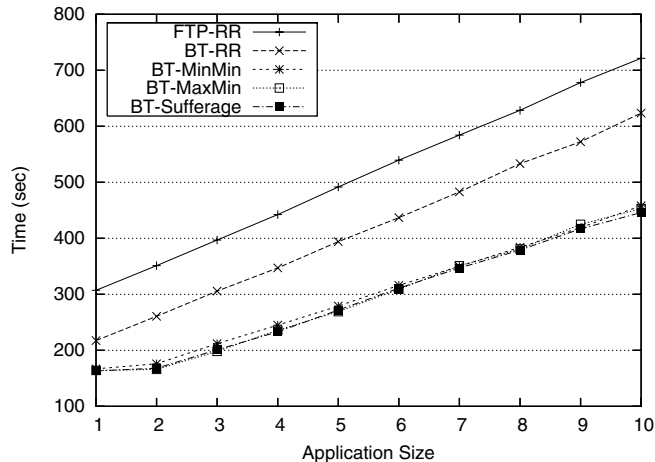


Fig. 8. Evaluation of scheduling heuristics according to Application Size.

3) *Application Computation/IO File Ratio*: Next experiment investigates the impact of communication/computation ratio on the application scheduling performance. In Figure 9 we observe that heuristics performs similarly when the CIOR increases.

When comparing the three knowledge-based heuristics, we can observe that MaxMin performs slightly better than MinMin and Sufferage, while the three heuristics perform similarly in the previous experience. Overall, it is hard to conclude about the relative efficiency of the three heuristics. There might be an opportunity for better performance forecasting than predictive communication ordering for instance by applying the NWS oracles [25].

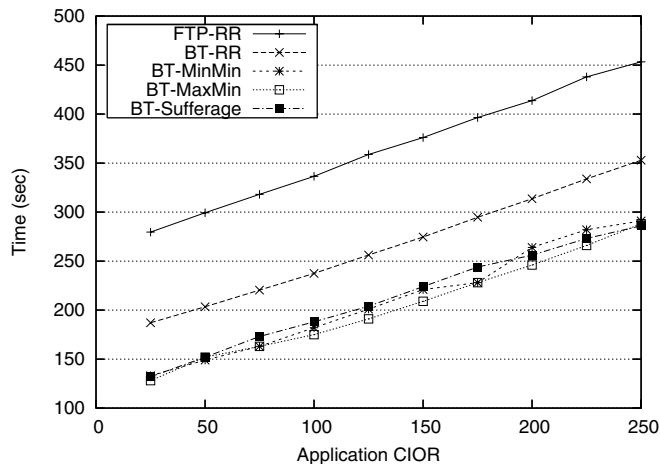


Fig. 9. Evaluation of scheduling heuristics according to Application CIOR.

4) *Application Share Data Ratio*: In the following, 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 SDR the difference ratio between the files. Figure 10 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 ($d = 100\%$).

<i>SDR</i>	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. 10. Effect of file sharing amongst tasks on BitTorrent distribution time to distribute a set of files to 20 nodes according to the difference ratio.

VI. CONCLUSION

Even simple scheduling heuristics indubitably benefit from collaborative data distribution over centralized data solution to execute data-centric parameter-sweep applications, onto medium to large collections of nodes. Nevertheless, our performance analysis of the BitTorrent protocol indicates that BitTorrent adds two constraint when designing efficient scheduling heuristics: the high overhead when distributing small files, and the unpredictability of communication patterns. Relative to these issues, our contribution is two fold:

- We have succeeded in designing a performance model of BitTorrent which accurately select the best of the two protocols according to the size of the file to distribute and the number of receiver nodes.
- We proposed an enhancement of the built-in incentive mechanism of BitTorrent in order to implement predictive and deterministic communication ordering. By constraining the BitTorrent protocol to accomplish the file transfer in a pre-determined sequence, we were able to calculate a prediction of the communication cost. Therefore, we have proposed and evaluated BitTorrent dedicated variants of well-known scheduling heuristics **BT-MinMin**, **BT-MaxMin** and **BT-Sufferage**. We obtain a speed-up of 3 when compared to classical round robin algorithm with the FTP protocol **FTP-RR**, and a speed-up of 1.5 when compared to round robin algorithm with the BitTorrent protocol **BT-RR**.

REFERENCES

- [1] I. Foster, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations, IISA, 2001," in *International Journal on Supercomputer Applications*, 2001.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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.

- [7] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.
- [8] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," in *Proceedings of IEEE INFOCOM*, March 2004.
- [9] 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.
- [10] EDonkey, "Edonkey, overnet homepage," January 2002, <http://www.edonkey200.com/>.
- [11] FastTrack, "P2P Technology. KaZaA Media Desktop," January 2002, <http://www.fasttrack.nu/>.
- [12] 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.
- [13] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proceedings of IEEE/INFOCOM 2005*, Miami, USA, March 2005.
- [14] 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.
- [15] B. Wei, G. Fedak, and F. Cappello, "Collaborative Data Distribution with BitTorrent for Computational Desktop Grids," in *The 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, Lille, France, July 2005.
- [16] M. Faerman, A. Su, R. Wolski, and F. Berman, "Adaptive Performance Prediction for Distributed Data-Intensive Applications," in *Proceedings of Supercomputing 1999*, I. C. S. Press, Ed., 1999.
- [17] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Bandwidth-centric allocation of independent tasks on heterogeneous platforms," in *International Parallel and Distributed Processing Symposium IPDPS'2002*. IEEE Computer Society Press, 2002.
- [18] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrante, "Autonomous Protocols for Bandwidth-Centric Scheduling of Independent-task Applications," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, April 2003.
- [19] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid environments," in *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'2000)*, Cancun, Mexico, 2000.
- [20] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids," in *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.
- [21] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 1995.
- [22] 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.
- [23] A. Bellissimo, P. Shenoy, and B. N. Levine, "Exploring the Use of BitTorrent as the Basis for a Large Trace Repository," University of Massachusetts, Tech. Rep., 2004.
- [24] A. R. Bharambe, H. Cormac, and V. N. Padmanabhan, "Understanding and Deconstructing BitTorrent Performance," Microsoft Research, Tech. Rep., 2005.
- [25] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Journal of Future Generation Computing Systems*, vol. 15, no. 5-6, pp. 757–768, October 1999.
- [26] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Heterogeneous Computing Workshop*, 1999, pp. 30–. [Online]. Available: citeseer.csail.mit.edu/maheswaran99dynamic.html
- [27] 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.