# freeCycles - Efficient Data Distribution for Volunteer Computing

Rodrigo Bruno Paulo Ferreira
INESC-ID - Instituto Superior Técnico, ULisboa
rodrigo.bruno@tecnico.ulisboa.pt, paulo.ferreira@inesc-id.pt

## Abstract

Volunteer Computing (VC) has been proving to be a way to access large amounts of computational power, network bandwidth and storage. With the recent developments of new programming paradigms and their adaptation to run on the large scale Internet, we believe that data distribution techniques need to be re-thought in order to cope with the high volumes of information handled by, for example, MapReduce. Thus, we present a VC solution called freeCycles, that supports MapReduce jobs. freeCycles presents two new contributions: i) improves data distribution (among mappers and reducers) by using the BitTorrent protocol to distribute (input, intermediate and output) data, ii) improves intermediate data availability by replicating it through volunteers in order to avoid losing intermediate data and consequently preventing big delays on the MapReduce execution time.

***Categories and Subject Descriptors*** C.2.2 [*Distributed Systems*]: Client/Server; C.2.5 [*Local and Wide-Area Networks*]: Internet; C.4 [*Performance of Systems*]: Reliability, availability, and serviceability

***Keywords*** Data Distribution, Volunteer Computing, BitTorrent, MapReduce

## 1. Introduction

With the ever growing demand for computational power, scientists all over the world strive to harvest more computational resources to solve bigger problems in less time. We believe that volunteer computing (VC) is an interesting solution for accessing large amounts of computational power. With time, more computing devices join the network. By aggregating all the resources in a global volunteer pool, it is possible to achieve a huge amount of computational power that would be impossible, or impractical, in most grids and clusters.

The creation and development of large scale VC systems will allow running large scale projects that could not be run on grids or clusters due to its size and costs associated. VC will also provide a platform to explore and create new projects without loosing time preparing and setting up execution environments like clusters. In addition, with the recent developments and applicability of the MapReduce [7] programming paradigm to VC, comes also the need to adapt and evolve current data distribution techniques to this new paradigm.

freeCycles is a VC solution and thus, its ultimate goal is to aggregate as many volunteer resources as possible to perform general purpose computation. As that being, our solution must be capable of scaling up with the number of volunteers and collecting volunteers' resources such as CPU cycles and network bandwidth in an efficient way. We also believe that being able to tolerate volunteer faults and unreliable network connections is an important requirement for our solution since volunteer node churn is very high in large scale networks (as the Internet). freeCycles must be capable of supporting new parallel programming paradigms, in particular MapReduce, given its relevance for a large number of applications. Finally, as MapReduce applications depend on large amounts of information to run, freeCycles must be capable of distributing large amounts of data (namely input, intermediate and output data) efficiently without compromising its scalability.

In order to understand the challenges inherent to building a solution like freeCycles, it is important to note that our project is targeted to volunteer pools (set of regular desktop computers). As opposed to clusters and grids, nodes within a volunteer pool have slow connection (high latency and low bandwidth) between each other, high node churn, resources focused on user tasks, and cannot be trusted.

By looking at all the available solutions that could fulfill our goal, note that solutions based on clusters and/or grids do not fit our needs; these are designed for controlled environments where node failure is very improbable, nodes are typically well connected to each other, nodes can be trusted and are very similar in terms of software and hardware. Having this in mind, solutions like HTCondor [18], Hadoop [21], XtremWeb [9] and other similar computing platforms are of no use to attain our goals.

When we turn to VC, we observe that most of existing solutions are built and optimized to run Bag-of-Tasks applications. Therefore, solutions such as BOINC [2], GridBot [16], Bayanihan [15] and many others [1, 3, 4, 13] do not sup-


*CloudDP'14*, April 13, 2014, Amsterdam, Netherlands.

http://dx.doi.org/10.1145/2592784.2592788

port the execution of MapReduce jobs, which is one of our requirements.

The few solutions that support MapReduce jobs still use naive data distribution techniques. Large input files are commonly downloaded from servers to clients in order to start the computation. Even if multiple clients download the same input information (which is the normal procedure when tasks are replicated), these file transfers are done through the central server. The problem is even worse when we consider paradigms like MapReduce where huge input and intermediate outputs must be distributed to several nodes (mappers and reducers). In particular, the SCOLARS [6] system already supports inter-client data transfer for intermediate outputs. However, all file transfers (input, intermediate and output) are still a point-to-point transfer (i.e. from one source to one destination).

In order to support the execution of a programming paradigm as MapReduce, that needs large amounts of data to process, new distribution techniques need to be employed.freeCycles is a VC platform, compatible with BOINC, that enables the deployment of MapReduce applications. It is based on the SCOLARS system that, besides supporting MapReduce jobs, also enables inter-client transfers. freeCycles goes one step further by allowing clients to help distributing the input, intermediate output, and final output data (through the BitTorrent[1] protocol and not point-to-point). Since task replication is always needed to prevent stragglers from delaying the whole computation and to detect false results, multiple clients will need the same input and will produce the same output. Thereby, multiple clients can send in parallel multiple parts of the same file to other clients or to the central server. freeCycles will, therefore, benefit from clients' network bandwidth to distribute data to other clients or even to the central server.

By providing this functionality, freeCycles achieves higher scalability (reducing the burden on the central server), reduced transfer time (improving the overall turn-around time) and augmented fault tolerance (since nodes can fail during the transfer without compromising the data transfer of other nodes).

The remainder of this document is organized as follows. Section 2 is dedicated to related work. Section 3 describes freeCycles' architecture. In Section 4, we evaluate our solution. Section 5 concludes the paper with some final remarks.

## 2. Related Work

Over time, many VC platforms have been developed but, within this section, we focus our analysis to the ones we think that are closer to our goals.

On one hand, solutions like Bayanihan [15], BOINC [2], and GridBot [16], among others, are mainly built and optimized for Bag-of-Tasks (embarrassingly parallel) applications and thus, cannot tolerate MapReduce applications. On the other hand, solutions like SCOLARS, MOON [12], and the solutions presented by Tang [17] and Marozzo [8] already support MapReduce applications.

MOON (MapReduce On Opportunistic Environments) is an extension of Hadoop (an open source implementation of MapReduce). MOON ports MapReduce to grid environments mainly by: i) modifying both data and task scheduling, and ii) performing intermediate data replication. However, MOON was designed to run on grids and not on volunteer pools and thus, many of its assumptions do not hold (e.g. having dedicated nodes for storage).

The solution presented by Tang [17] is also an implementation of MapReduce focused on grids. It was built on top of a data management framework, Bitdew [10]. We believe that this framework would produce high overhead in a VC setting and it assumes high availability of a particular set of nodes (which is prohibitive in a volunteer pool).

Marozzo [8] presents a solution to exploit the MapRduce model in dynamic environments. The major drawbacks of this solution are: i) data is distributed point-to-point (which fails to fully utilize the volunteer's bandwidth), and ii) there is no intermediate output replication.

SCOLARS (Scalable Complex Large Scale Volunteer Computing) is a VC platform built on top of BOINC. It presents two main contributions: i) inter-client transfers (for intermediate output), and ii) hash based task validation (where only a hash of the output is validated on the central server). However, it presents the same issues as the previous solution: only point-to-point transfers and no intermediate data replication.

Regarding data distribution systems, we analysed several but we only discuss the ones we think that are relevant to our objective (distribute efficiently large amounts of data within a volunteer pool): FastReplica [5] and BitTorrent [14].

FastReplica is a replication algorithm focused on replicating files in Content Distribution Networks (CDNs). It was designed to be used in large-scale distributed network of servers and it uses a push model (where the sender triggers the data transfer). FastReplica works in two steps: i) distribute equally sized pieces of the original data among the destination servers; ii) all destination servers send to all other destination servers their piece. Despite being very efficient, we point out two main issues: i) it relies on a push model, which can be very difficult to use when there is a variable set of destination nodes; ii) it does not cope with node failure (since it was designed for servers, which are supposed to be up almost all the time).

BitTorrent is a peer-to-peer data distribution protocol widely used to distribute large amounts of data over the Internet. It was designed to avoid the bottleneck of file servers (like FTP servers). BitTorrent can be used to reduce the server and network impact of distributing large files by allowing ordinary users to spare their upload bandwidth to spread the file. So, instead of downloading the whole file from a server, a user can join a swarm of users that share a particular file and thereafter, download and upload small file chunks from and to multiple users until the user has all the file chunks.

[1] Official BitTorrent Specification can be found at www.bittorrent.org

Using this protocol it is possible to use computers with low bandwidth connections and even reduce the download time (compared to a centralized approach).

In order to find other users sharing the target file, one has to contact a BitTorrent Tracker. This tracker has to maintain some information about the nodes such as a node's ip, port to contact and available files. This information is kept so that when a node asks for a file, the tracker is able to return a list of nodes to whom the node should contact.

This protocol has been proving to be capable of scaling up to millions of users and providing high efficiency in distributing large amounts of data [14]. Thus, we intend integrate this protocol in our solution since it enables a pull model where data can flow as volunteers need it (as opposed to FastReplica).

## 3. freeCycles

freeCycles is a volunteer computing solution that strives to improve the data distribution efficiency and reliability which are specially needed to run new parallel programming paradigms, MapReduce in particular, on large scale volunteer pools. Our solution also takes into account another important factor: intermediate data availability. As has been shown [11], the loss of intermediate data can incur into a large overhead within a MapReduce workflow.

freeCycles focuses on improving the data distribution techniques; therefore, scheduling, task validation, and other issues associated with implementing a complete volunteer computing platform, are out of the scope for this project. Hence, we decided to use an existing system, SCOLARS, as a base platform for our work. SCOLARS is a recent project based on BOINC (to our knowledge, the most successful VC platform) that introduces the MapReduce paradigm to volunteer computing, making it possible to run MapReduce jobs over large volunteer pools. Besides bringing MapReduce to volunteer computing, SCOLARS supports two interesting features: i) inter-client communication to transfer intermediate outputs from mappers to reducers; ii) a hash based task validation mechanism.

Regarding the data distribution protocol (which is used to move data between clients and servers) we decided, based on our analysis of current solutions, to use the BitTorrent protocol. Furthermore, BitTorrent was tested against other protocols and the results showed that it was considerably more scalable and more efficient than the other approaches [19, 20]. Also, recent data from Ipoque, an Internet traffic management and analysis company[2], shows that, nowadays, BitTorrent represents 27% to 55% of all the Internet traffic (depending on geographical location).

### 3.1 Architecture

In this section, we describe freeCycles architecture for both server and client sides. Some of its basic components are already present in SCOLARS. We focus on the freeCycles extensions.

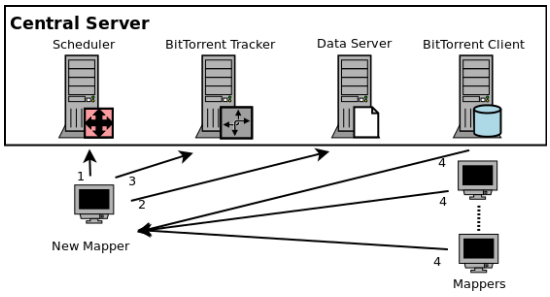


**Figure 1.** Input Data Distribution

The basic server-side architecture is composed by several components: a database where information about jobs is held; a data server where input and output data is stored and a scheduler that is responsible for delivering tasks to volunteers when asked. freeCycles adds two additional components (see Figure 1): a BitTorrent tracker (to enable volunteers to use the BitTorrent protocol to download and upload data to and from clients and the central server), and a BitTorrent client (that will be used to share the initial input and to receive the final output through the BitTorrent protocol).

With respect to the client-side architecture, freeCycles adds a BitTorrent client to enable BitTorrent file transfers between the server and the client, and also between clients.

By using such BitTorrent clients and a tracker, freeCycles is able to move data between clients and server taking advantage of the download and upload bandwidth available at the volunteer nodes. This imposes a significant decrease in the used bandwidth and load of the central server.

### 3.2 Data Distribution Algorithm

Having described the components on (client-side) volunteers and on the server, we now detail how we use the BitTorrent file sharing algorithm to coordinate input, intermediate and output data transfers.

#### 3.2.1 Input Distribution

Input distribution is the first step in every MapReduce application. Input must be split over multiple mappers. To do so, each mapper downloads a .torrent file pointing to the corresponding input file from the central data server.

For each input file, the server play as initial seed (the origin). If we take into consideration that each map task is replicated over at least three volunteers (for validation purposes), then, when a new map task begins, the volunteer will have at least one seed (the server) and possibly one or two additional volunteers sharing the file (each volunteer shares the input file using the BitTorrent protocol).

Therefore, we can leverage the task replication mechanisms to share the burden of the data server. Even if the server is unable to respond, a new mapper may continue to download its input data from other mappers. The transfer band-

[2] www.ipoque.com

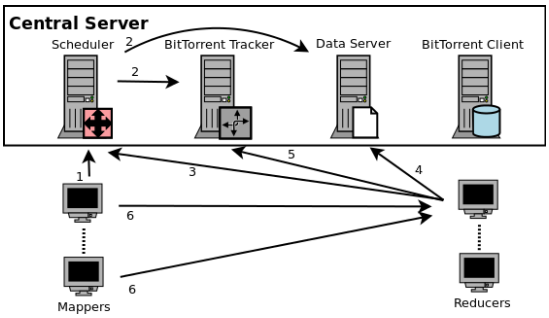


**Figure 2.** Intermediate Data Distribution

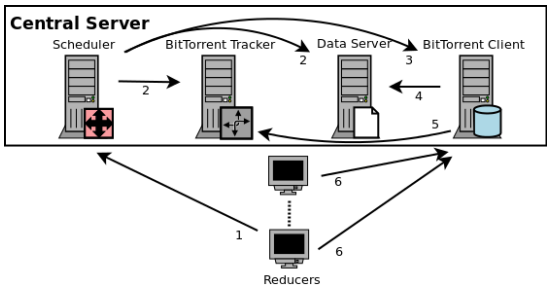


**Figure 3.** Output Data Distribution

width will also be bigger since a mapper may download input data from multiple sources.

Input data distribution is done as follows (see Figure 1): 1) a new volunteer requests work from the central server scheduler and receives a workunit; 2) the new mapper downloads the .torrent file from the data server (a reference of the .torrent file was inside the workunit description file); 3) the new mapper contacts the BitTorrent tracker to know about other volunteers sharing the same data; 4) the volunteer starts downloading input data from multiple mappers and/or from the server.

#### 3.2.2 Intermediate Output Distribution

Once a map task is finished, the mapper has an intermediate output ready to be used. The first step is to make the server aware of the map task finish. To this end, the mapper contacts the server and sends a hash of the intermediate output that will be used to validate the task output.

As more hashes arrive at the server, the server is able to decide which peers (mappers) have the correct intermediate files. This information is placed at the BitTorrent tracker and a new .torrent file is created publishing where future volunteers playing the reduce role can find this intermediate output. When all the intermediate outputs are available, the scheduler starts issuing reducer tasks. These reducer tasks have a reference to the .torrent files that need to be downloaded. Once a reducer has access to these .torrent files, it starts transferring the intermediate files (using the BitTorrent protocol) from all the mappers that completed the map task with success (the ones that were successfully validated). Reduce tasks start as soon as all the needed intermediate values are successfully transfered.

Intermediate data distribution works as follows (see Figure 2): 1) a message acknowledging the map finish and containing the computed intermediate output hash is sent to the scheduler; 2) the central server produces a .torrent file and stores it in the data server and information about how to access the mappers holding the data is stored in the BitTorrent tracker; 3) when a new volunteer (reducer) asks for work, a workunit is delivered with a reference to a .torrent file; 4) the reducer downloads the .torrent file from the data server; 5) the reducer contacts the BitTorrent tracker to know about the mapper nodes that hold the data; 6) the reducer uses its BitTorrent client to download the intermediate data from multiple mappers.

#### 3.2.3 Output Distribution

Given that map and reduce tasks are replicated over at least three volunteers, it is possible to accelerate the upload of the final output files from the reducers to the data server.

The procedure is similar to the one used for intermediate outputs. Once a reduce tasks finishes, it contacts the server providing a hash of the final output. The server save hashes from multiple reducer replicas and decides which ones are the trustworthy replicas. With this information, the server updates the tracker and creates a .torrent for each one of the reducer's outputs. These .torrent files will be consumed by the BitTorrent client placed at the server. Therefore, after the first reduce task validation finishes, the server can start downloading the final output from multiple volunteer nodes.

Using BitTorrent to transmit the final outputs results in a faster transfer from volunteers to the data server, a lower and shared bandwidth consumption from the volunteer's perspective, and an increased fault tolerance (since a volunteer node failure will not abort the file transfer).

Output data distribution works as follows (see Figure 3): 1) a message acknowledging the reduce finish and containing the computed output hash is sent to the scheduler; 2) a .torrent file is created and stored in the data server, the BitTorrent tracker is updated (with the reducer access information); 3) the BitTorrent client is notified; 4) the BitTorrent client inside the central server downloads the .torrent file from the data server; 5) the BitTorrent client contacts the BitTorrent tracker to know how to reach the volunteers holding the output; 6) the BitTorrent client downloads the output data from multiple reducers.

### 3.3 Availability of Intermediate Data

Previous studies [11] show that the availability of intermediate data is a very sensitive issue in programming paradigms like MapReduce. This problem takes even bigger proportions when applied to volatile environments like volunteer pools. To cope with this problem, freeCycles presents two methods.

First, replicate map tasks aggressively when volunteers designated to execute a particular map task take too long to report results. By imposing a shorter interval time to respond with results, we make sure that we keep at least a few replicas of the intermediate output.

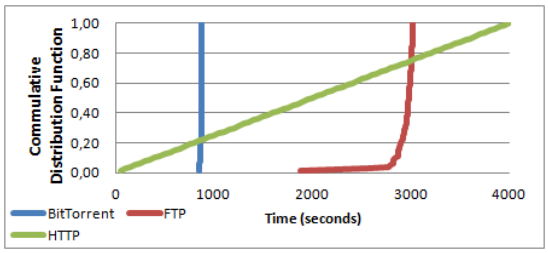


**Figure 4.** CDF for input distribution

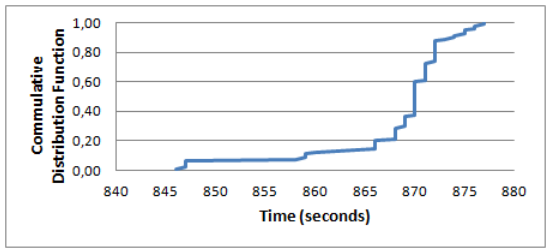


**Figure 5.** CDF for BitTorrent during input distribution

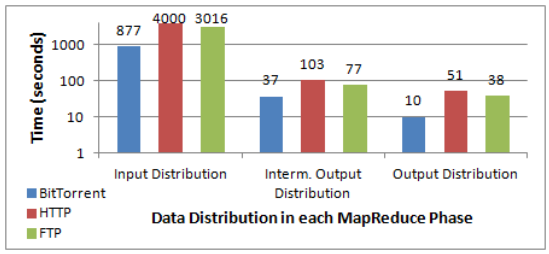


**Figure 6.** Performance evaluation of the protocols used in BOINC (HTTP), SCOLARS (FTP) and freeCycles (BitTorrent)

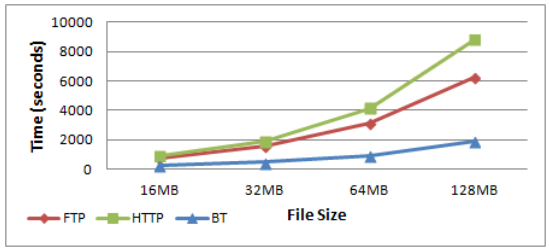


**Figure 7.** Performance comparison while varying the input file size

Second, when there are intermediate outputs that have already been verified (by the central server) and other map tasks are still running, new tasks designed to replicate intermediate output might be delivered to new volunteers. Therefore, volunteers might be used to replicate intermediate data to compensate other mappers that might die while waiting for the reduce phase to start. These tasks would simply download .torrent files and use them to start downloading intermediate data.

However, while fighting for the availability of intermediate data, we might be spending valuable resources that could be used for advancing towards the job completion. For example, if we use too much bandwidth replicating intermediate data, less bandwidth will be available to send intermediate data to reducers. Accordingly, we expect to be able to experiment and to end up with a good tunning of both weights.

## 4. Evaluation

In this section we present a performance comparison between the data distribution protocols used in BOINC (HTTP), SCOLARS (FTP) and freeCycles (BitTorrent) in the context of a MapReduce application. All three protocols were evaluated using a simulated MapReduce data flow comprehending three phases: i) Input Distribution: central server sends initial input to all mappers; ii) Intermediate Output Distribution: mappers send intermediate outputs to reducers; iii) Output Distribution: reducers send final output to the central server.

All tasks (maps and reduces) were replicated: map tasks with a replication factor of 5 (high replication factor to simulate aggressive replication in order to keep intermediate data available); reduce tasks with a replication factor of 3 (minimum for majority voting).

We used a total of 92 nodes (92 cores with 2GB of RAM, scattered in several physical nodes): 80 mappers (16 different map tasks replicated 5 times) and 12 reducers (4 different reduce tasks replicated 3 times). To simulate Internet connection bandwidths, all nodes had their download and upload bandwidths limited to 100Mbps and 10Mbps respectively (except for the experiment in Figure 8). All the files (16 input files, 16 intermediate outputs and 4 final outputs) were different with 64MB each (except for the experiments in Figures 7 and 8).

Figure 4 presents the Cumulative Distribution Function (CDF) for all the three protocols during input distribution. It is clear that BitTorrent is able to distribute the same amount of data much faster than FTP and HTTP. From Figure 5 (that shows a zoomed region of Figure 4), we see why BitTorrent is able to finish all the transfers earlier: as soon as one mapper has some part of the file, it can participate in the input distribution. Therefore, the more mappers have the input file (or at least some parts), the faster it gets to other mappers download the file. We do not present the CDFs for intermediate output distributed since results are similar.

From Figure 6, we can conclude that the BitTorrent protocol outperforms the other two protocols in all the data distribution phases. If we take into consideration that SCOLARS only uses FTP to distribute intermediate data, and still uses HTTP to distribute input and output data, we achieve a total of 4128 seconds spent distributing data. Looking at freeCycles, with BitTorrent distributing all data, we achieve a total of 924 seconds: freeCycles reduces the time spent data by 77,6% compared to SCOLARS.

We further analysed the behaviour of the three protocols while varying: i) the file input size, and ii) the upload bandwidth (for a constant download bandwidth). In the first scenario (Figure 7) we repeated the experiment in Figure 6 for

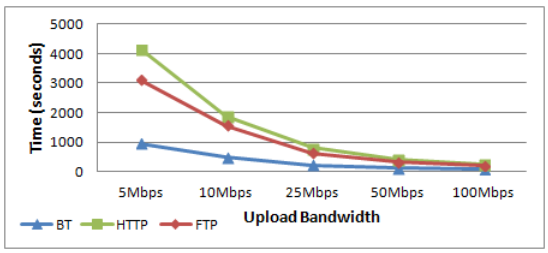


**Figure 8.** Performance comparison while varying the node upload bandwidth

different file sizes. The second scenario (Figure 8) is similar to the first one but, instead of varying the file size, we used different upload bandwidths (keeping the download bandwidth at 100Mbps). For both simulations, we only present the sum of the time it took to distribute all data (input, intermediate and output). It is then possible to conclude that BitTorrent is more scalable while increasing the file size and decreasing the upload bandwidth (which is an important result since ISPs provide much lower bandwidth for upload than for download).

## 5. Conclusions

In this paper we described the freeCycles VC platform. It brings two new contributions: 1) improved input, intermediate and final output data distribution with the BitTorrent protocol; 2) improved task scheduling that strives to keep intermediate data available.

Although freeCycles is still under active development we present a performance comparison of the three protocols used so far in a MapReduce data flow environment. We conclude that the proposed protocol (BitTorrent) outperforms other protocols thus motivating even more the development of freeCycles.

#### Acknowledgements

This work was partially supported by national funds through FCT - Fundação para a Ciência e Tecnologia, under projects PTDC/EIA-EIA/113993/2009 and PEst-OE/EEI/LA0021/2013.

## References

[1] A. Alexandrov, M. Ibel, K. Schauser, and C. Scheiman. Superweb: towards a global web-based parallel computing infrastructure. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 100–106, 1997.

[2] D. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 2004.

[3] A. Baratloo, M. Karaul, Z. Kedem, and P. Wijckoff. Charlotte: Metacomputing on the web. *Future Generation Computer Systems*, 1999. ISSN 0167-739X.

[4] A. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 2005. ISSN 1083-4427.

[5] L. Cherkasova and J. Lee. Fastreplica: Efficient large file distribution within content delivery networks. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[6] F. Costa, L. Veiga, and P. Ferreira. Internet-scale support for map-reduce processing. *Journal of Internet Services and Applications*, 2013. ISSN 1867-4828.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, Jan. 2008. ISSN 0001-0782.

[8] D. T. Fabrizio Marozzo and P. Trunfio. Adapting mapreduce for dynamic environments using a peer-to-peer model, 2008.

[9] G. Fedak, C. Germain, V. Neri, and F. Cappello. Xtremweb: a generic global computing system. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 582–587, 2001.

[10] G. Fedak, H. He, and F. Cappello. Bitdew: A data management and distribution service with multi-protocol file transfer and metadata abstraction. *Journal of Network and Computer Applications*, 2009. ISSN 1084-8045.

[11] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. Making cloud intermediate data fault-tolerant. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 181–192. ACM, 2010.

[12] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang. Moon: Mapreduce on opportunistic environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 95–106, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-942-8.

[13] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *Peer-to-Peer Systems III*, pages 227–236. Springer, 2005.

[14] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[15] L. F. Sarmenta and S. Hirano. Bayanihan: building and studying web-based volunteer computing systems using java. *Future Generation Computer Systems*. ISSN 0167-739X.

[16] M. Silberstein, A. Sharov, D. Geiger, and A. Schuster. Gridbot: execution of bags of tasks in multiple grids. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 11:1–11:12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-744-8.

[17] B. Tang, M. Moca, S. Chevalier, H. He, and G. Fedak. Towards mapreduce for desktop grid computing. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*, pages 193–200, 2010.

[18] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 2005. ISSN 1532-0634.

[19] B. Wei, G. Fedak, and F. Cappello. Scheduling independent tasks sharing large data distributed with bittorrent. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 219–226. IEEE Computer Society, 2005.

[20] B. Wei, G. Fedak, and F. Cappello. Towards efficient data distribution on computational desktop grids with bittorrent. *Future Generation Computer Systems*, 2007.

[21] T. White. *Hadoop: the definitive guide*. O'Reilly, 2012.