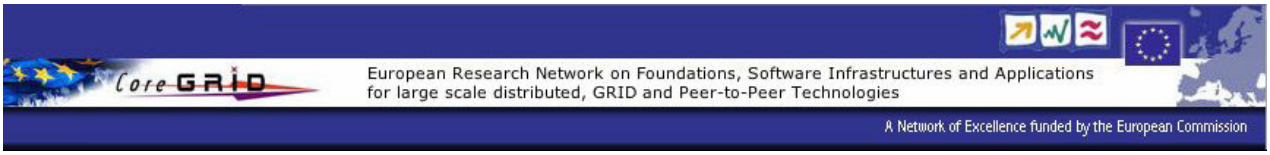

Peer-To-Peer Techniques for Data Distribution in Desktop Grid Computing Platforms

Fernando Costa[1], *Luis Silva*[1], *Ian Kelley*[2], *Ian Taylor*[3]

[1]*CISUC - Centre for Informatics and Systems of the University of Coimbra*
Polo II - Pinhal de Marrocos
3030-290 Coimbra, Portugal
luis@dei.uc.pt

[2]*Center for Computation & Technology*
Louisiana State University, United States
I.R.Kelley@cs.cardiff.ac.uk

[3]*School of Computer Science,*
Cardiff University, United Kingdom
Ian.J.Taylor@cs.cardiff.ac.uk

CoreGRID Technical Report
Number TR-0095
August 30, 2007

Institute on Architectural issues: scalability, dependability, adaptability (SA)

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-00426

Peer-To-Peer Techniques for Data Distribution in Desktop Grid Computing Platforms

Fernando Costa[1], Luis Silva[1], Ian Kelley[2], Ian Taylor[3]

[1]CISUC - Centre for Informatics and Systems of the University of Coimbra
Polo II - Pinhal de Marrocos
3030-290 Coimbra, Portugal
luis@dei.uc.pt

[2]Center for Computation & Technology
Louisiana State University, United States
I.R.Kelley@cs.cardiff.ac.uk

[3]School of Computer Science,
Cardiff University, United Kingdom
Ian.J.Taylor@cs.cardiff.ac.uk

CoreGRID TR-0095[*]
August 30, 2007

Abstract

In this paper, we discuss how Peer-to-Peer data distribution techniques can be adapted to Dekktop Grid computing environments, particularly to the BOINC platform. To date, Desktop Grid systems have focused primarily on utilizing spare CPU cycles, yet have neglected to take advantage of client network capabilities. Leveraging client bandwidth will not only benefit current projects by lowering their overheads but will also facilitate Destkop Grid adoption by dataheavy applications. In this paper, we outline two approaches to Peer-to-Peer data sharing that could be adapted for volunteer computing platforms: the highly successful BitTorrent protocol; and, a more complex, yet secure and customizable, hierarchal Peer-to-Peer data center approach.

[*] *This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).*

1 Introduction

Desktop Grids have been extremely successful in bringing large numbers of donated computing systems together to form computing communities with vast resource pools. These types of systems are well suited to perform highly parallel computations that do not require any interaction between network participants. Currently, the most successful Desktop Grid systems are volunteer computing platforms such as the Berkeley Open Infrastructure for Network Computing (BOINC), which rely on donated computer cycles from ordinary citizen communities. BOINC is currently being successfully used by many projects to analyze data, and with a supportive user community can provide compute power to rival that of the world's supercomputers. In the current implementation of these systems, network topology is restricted to a strict master/worker scheme, generally with a fixed set of centrally managed project computers distributing and retrieving results from network participants. The potentially large user communities that become involved in volunteer computing initiatives can easily result in large network requirements for host projects, forcing them to upgrade their computer hardware and network availability as their projects rise in popularity.

These centralized data architectures currently employed by BOINC and other Desktop Grid systems, can be a potential bottleneck when tasks share large input files or the central server has limited bandwidth. With new data management technologies, Desktop Grid users will be able to explore new types of data-intensive application scenarios – ones that are currently overly prohibitive given their large data transfer needs. This lack of a robust data solution often discourages application developers from embracing a Desktop Grid environment, or forces users to scale back their applications to only problems that do not rely upon large data sets. There are many applications that, given more robust data capabilities, could either expand their current problem scope, or migrate to a Desktop Grid environment.

Peer-to-Peer (P2P) data sharing techniques can be used to introduce a new kind of data distribution system for volunteer and Desktop Grid projects – one that takes advantage of client-side network capabilities. This functionality could be implemented in a variety of forms, ranging from BitTorrent-style networks where all participants share equally, to more constrained and customizable unstructured P2P networks where certain groups are in charge of data distribution and discovery. These approaches, although similar in nature, each have their own distinct advantages and disadvantages, especially when considered in relation to a scientific research community utilizing volunteer resources. In this paper, we make the argument for P2P data distribution, discuss the relative advantages and disadvantages of these two approaches, and explore how they could be applied to the Desktop Grid community, with particular emphasis on BOINC. This paper is organized as follows: section 2 gives background on the technologies involved; section 3 introduces related work; section 4 discusses how P2P technologies could be applied to Desktop Grid systems such as BOINC; section 5 introduces how the BitTorrent protocol could be used in this facility; section 6 presents a more complex data center approach; and, section 7 concludes.

2 Background

To begin the discussion on how P2P technologies can be integrated into Desktop Grids, and specifically BOINC, it is advantageous to first give a brief overview of the software technologies involved. Naturally, there are many [3] [15][22] Peer-to-Peer technologies available, and several different systems that can be classified as Desktop Grids [1][6][12][16], however, for the purposes of this paper, we are going to limit our scope to exploring how the very popular BitTorrent protocol, as well as another in-development secure data center approach can both be applied to the most widespread "volunteer computing" Desktop Grid platform, the Berkeley Open Infrastructure for Network Computing (BOINC).

2.1 BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC) [1][4] is a software platform for distributed computation using otherwise idle cycles from volunteered computing resources. BOINC's use is widespread, with many different and varying projects employing the core infrastructure to distribute their data processing jobs. The diverse scientific domains utilizing BOINC range from gravitational wave analysis, to protein folding, to the search for extraterrestrial life[†]. Although these projects are diverse in their scientific nature, each one has something in common with the others: they have work units that can be easily distributed to run autonomously in a highly distributed and volatile environment. To achieve this task, each project must not only prepare its data and executable code to work with the BOINC libraries and client/server infrastructure, but they must also setup and maintain their

[†] See a comprehensive list of BOINC projects at: http://boinc.berkeley.edu/projects.php

own individual servers and databases to manage the project's data distribution and result aggregation. BOINC has been highly successful, and to date, over 5 million participants have joined various BOINC projects, giving an overall computing power equivalent to 450 TeraFlops [2].

2.2 BitTorrent

BitTorrent [7] is a popular file distribution protocol based on the P2P paradigm. However, unlike other well-known P2P applications such as Gnutella or KaZaA, which incorporate peer and file discovery algorithms, BitTorrent's focus is more on optimising the distributed of files by enabling multiple download sources through the use of file partitioning, tracking and file swarming techniques. The main idea of BitTorrent is the collaboration between users accessing the same file by sharing chunks of the file with each other. To obtain information about the file to download, a peer must download a corresponding .torrent file. This file contains the file's length, name and hashing information, and the url of a tracker, which keeps a global registry of all the peers sharing the file. Trackers help peers establish connections between themselves by responding to a user's file request with a partial list of the peers having (parts, or chunks of) the file. A tracker does not participate in the actual file distribution, and each peer decides locally which data to download based on data collected from its neighbors. Therefore, each peer is responsible for maximizing its own download rate. Peers do this by downloading from whoever they can and deciding which peers to upload to via a variant of tit-for-tat to prevent parasitic behavior. To cooperate, peers upload (unchoke in BitTorrent terminology), and its dual operation is choking, a temporary refusal to upload. A BitTorrent peer always unchokes a fixed number of peers (normally four), so it has to decide which peers to unchoke. The protocol follows two rules: 1) selection of which peers to unchoke is solely based on current download rate, and 2) optimistic unchoking is executed every 30 seconds, which consists of uploading to a fifth random peer.

2.3 P2P-ADICS

The Peer-to-Peer Architecture for Data-Intensive Cycle Sharing (P2P-ADICS) [17] is a research and development project at Cardiff University, working to build a multi-purpose and adaptable super-peer architecture for data caching that can be used by scientific applications to distribute large data files and large data sets in Desktop Grid environments. P2P-ADICS's is being designed with the scientific user in mind, taking into account such issues as customizable network membership and data security policies, as well as the more traditional scalability challenges. For its low-level network building layer, P2P-ADICS is currently relying on a software package entitled "Peer-to-Peer Simplified," or P2PS [20], which is also being developed by the same group. P2PS is a light-weight system for building decentralized Peer-to-Peer networks, and is similar in nature to JXTA, however it is more focused on the fundamental network building tools and provides much simpler mechanisms for advertisement queries and service discovery. In addition to basic network communications, P2PS provides the ability to bind to "virtual endpoints," which allows for endpoint-to-endpoint channels to be setup over different transport protocols (e.g., TCP or UDP). P2PS can be used by a variety of applications to construct P2P overlay networks, for a variety of purposes, including data exchange and caching.

3 Related Work

The creation of Condor [16], as one of the first Grid Computing middleware projects, paved the way for numerous Desktop Grid projects, that, instead of harnessing computational power from clusters on organizations, sought to take advantage of the internet and distributed desktop users. Many of these projects follow a centralized architecture [1][6][19], using a data distribution system that has one (or few when using mirrors) point of failure. To distribute data sharing, numerous alternatives are available today, in the form P2P file sharing systems or data storage systems. In this section we discuss some of the more significant ones as they relate to the work proposed here.

OceanStore [11] is a global, distributed, Internet-based storage infrastructure. It consists of cooperating servers, which work as both server and client. The data is split up in fragments which are stored redundantly on the servers. For search, OceanStore provides the Tapestry [18] subsystem, and updates are performed by using Byzantine consensus protocol. This adds an unnecessary overhead since file search is not a requisite for BOINC, and supporting replication implies the use of a distributed locking service, which incurs further performance penalties. Farsite also uses the Byzantine agreement protocol to establish trust within an untrusted environment. Farsite aims to provide the user with persistent non-volatile storage with a filesystem like interface, by utilizing unused storage from user workstations, whilst operating within the boundaries of an institution.

Frangipani [24] is a performance oriented Distributed Storage System typically used by applications which require a high level of performance. It follows a server-client architecture, and was implemented on top of the Petal system, employing Petal's low-level distributed storage services. It is designed to be utilized within the bounds of an institution where servers are assumed to be connected by a secure high bandwidth network, which goes against the global distribution of BOINC. Furthermore, like OceanStore, Frangipani also implements a distributed locking service, causing a considerable performance drop when servers access the same file.

Freeloader [18] aggregates unused desktop storage space and I/O bandwidth into a shared cache/scratch space, for hosting large, immutable datasets and exploiting data access locality. It is designed for large scientific results (outputs of simulations. The overall architecture of Freeloader shares many similarities to Google File System (GFS) [9]. GFS is a distributed storage solution which scales in performance and capacity whilst being resilient to hardware failures. GFS was designed to operate in a trusted environment, where the application is the main influence of usage patterns. The GFS typical file size was expected to be in the order of GB's and the application workload would consist of large continuous reads and writes, which does not apply to the BOINC environment.

Gnutella [10] is a decentralized file-sharing system whose participants form a virtual network, communicating via the Gnutella protocol, which is a simple protocol for distributed file search. To participate in Gnutella a peer first must connect to a known Gnutella host (host lists are available on specialized sites). The search queries are broadcasted over the network, which cause very high bandwidth consumption, and no reputation system exists making it impossible to establish a trust-based mechanism.

KaZaA [15] is another decentralized file-sharing system. It was one of the first P2P systems to exploit peer heterogeneity by organizing the peers into two classes, Super Nodes (SNs) and Ordinary Nodes (ONs). SNs are generally more powerful in terms of connectivity, bandwidth, processing, and non-NATed accessibility. Each ON has a parent SN, which maintains a local index for the files of all of its children ON, creating a distributed file index. To bypass firewall and NATs, KaZaA uses KaZaA uses dynamic port numbers along with its hierarchical design - a node acts as a TURN server between 2 other nodes, relaying the communication between them. Like Gnutella, its file discovery mechanism creates unnecessary traffic, and the Super Node architecture applied to data distribution on BOINC would generate an intolerable amount of traffic in the relaying nodes.

4 Applying a Peer-to-Peer Data Architecture to BOINC

The BOINC architecture is based on a strict master/worker model, with a central server responsible for dividing applications in thousands of small independent tasks and then distributing the tasks to participants, or worker nodes, as they request work units. To simplify network communication and bypass any NAT problems that might arise with bidirectional communication, the centralized server never initiates communication with worker nodes, rather all communication is instantiated from the worker when more work is needed or results are ready for submission. In the current implementation of BOINC, data distribution and scaling is achieved though the use of multiple centralized and mirrored HTTP servers that share data with the entire network.

The centralized architecture of BOINC not only creates a single, or in the case of mirrored servers, small number of failure points and potential bottlenecks, but it also fails to take advantage of the client-side network bandwidth and capabilities. If client-side network bandwidth could be successfully utilized to distribute data sets, not only would it allow for larger data files to be distributed, but it would also minimize the needed network capabilities of BOINC projects, thereby substantially lowering operation costs. To decentralize the current model as it relates to data, we propose using a Peer-to-Peer data distribution approach.

When considering the practical application of P2P technologies to the "production" BOINC environment, several concerns must be adequately addressed if the solution is to be successful. For the purposes of this paper, we have chosen to focus on the following four:

- **Router Configuration** — a Peer-to-Peer infrastructure would have to have a way to automatically configure routers or somehow bypass NAT issues through use of relaying severs
- **Data Integrity** — mechanisms for identifying hosts that supply bad data, and subsequently banning them from the network or having ways to avoid using them
- **Adaptable Network Topology** — ability to not only adapt on on the wide area network, but also to detect and exploit local area network topologies and relative proximity
- **BOINC Integration** — any new technology must be easy to integrate with current BOINC client software, in practice this means a C++ implementation or binding

4.1 Case Study of Two Selected P2P Approaches

Applying a P2P data distribution approach could be achieved in a variety of forms. In this paper, we discuss two implementations: one that uses a centralized tracker, as in BitTorrent [§5] where worker-nodes each share data; and the other that employs the use of decentralized data servers [§6] built using a super-peer topology, which could be configured to limit data sharing participants based upon project defined security constraints. In the latter case, these policies could be implemented to have the data layer mimic the currently used system of a few known and trusted peers, yet would scale as the network size or data loads increase (by requesting more trusted peers to become data centers). Either of these types of systems would be especially beneficial to projects that: have large input files; use the same input file for several work units; and/or, have limited or slow outbound connections from the central project server. In the rest of the paper, we will present these two different approaches in more detail, and outline what they would require to be applied to a BOINC application.

5 Approach 1: Adapted BitTorrent for Data Distribution

In order to integrate BitTorrent in BOINC, the main BOINC server code remains relatively unchanged but a tracker is needed to co-ordinate the downloads. The tracker manages the .torrent file once it is created, and acts as the first seed in the network. On the client side, not only is a BitTorrent client needed to download and share the file, but changes to the BOINC client code would be required. This is due to several reasons but mainly concerned with the starting and stopping of the BitTorrent client, as well as handling its errors and managing its execution requirements, such as downloading and rebuilding files, verifying signatures, and removing obsolete .torrents.

There are some advantages and disadvantages to implementing a pure BitTorrent solution. The advantages are many, for example, BitTorrent: has proven itself to be an efficient and low-overhead means of distributing data; can scale easily to large numbers of participants; and has built-in functionality to ensure relatively equal sharing ratios [11]. Some of these advantages however turn into disadvantages when trying to apply BitTorrent to a volunteer computing platform. For example, because of its flat topology, BitTorrent only works if enough nodes in its network are listening for incoming connections, which can prove problematic when confronted with firewalls and NAT systems. Another potential disadvantage when applying BitTorrent to the volunteering computing platform its "tit-for-tat" sharing requirements, which forces most participants to share on a relatively equal scale to what they are receiving. Although this proves quite effective for preventing selfish file-sharing on traditional home networking systems, it is not necessarily a requirement when applying P2P technologies to volunteer computing. For example, in the volunteer computing case, not everyone may wish to be a BitTorrent node but they may wish to offer their CPU time to a project. So, in the pure tit-for-tat BitTorrent world, this would not be possible.

In the following, the four target issues identified earlier in §4 are discussed, with a brief overview of how they relate to BitTorrent intregration.

- **Firewall & Router Configuration** — BitTorrent, as other P2P protocols, is based on a two-way communication between peers. Every peer, seed or not, is supposed to accept requests for chunks from other peers, and therefore must allow incoming connections, by opening the BitTorrent port (usually in the 6881– 6889 range) in their routers/firewalls. In a BitTorrent swarm, should no peer accept incoming connections (including the initial seed), the system would not work.
  There is no easy answer for this problem, faced by most P2P protocols. If both clients are behind symmetric NATs, the only solution is to use a relay server, possibly a node with a public IP that would act as an intermediary between two clients. This methodology is used by Skype, but it would prove disastrous in this case, given the size of the shared files, causing an excessive overhead on the relay. For non-symmetric NATs, hole punching techniques could be used, but it would involve changes in the BitTorrent core software layer, which is beyond the scope of this paper.
- **Malicious Users** — The integration of BitTorrent would bring new security issues to BOINC, and creates more possibilities for malicious users to exploit the system. The BitTorrent protocol itself does not strictly enforce fairness and exploits are possible, but the use of a central tracker decreases the danger of malicious attacks. Hashing prevents bad data from being propagated across the network, and small chunk sizes can be used to avoid downloading too much corrupted data. An additional level of security is provided by certain BitTorrent clients like Azureus[‡], that bans peers that share bad data. The "original"

[‡]See project web site at: http://azureus.sourceforge.net/

BitTorrent client by Bram Cohen [7] also incorporates a similar mechanism by default, with the tag –retaliate to garbled data, which refuses further connections from addresses with broken or intentionally hostile peers.

Therefore, the main problem with BitTorrent is not in the protocol itself, but rather in the peer swarms which allow BOINC users to obtain a list of other users that are downloading the same file (and possibly executing the same work unit). A client could send consecutive requests for peer lists to the tracker, and build a comprehensive database of peers sharing a file. Should a user from the list answer the attacker and agree to cooperate with him, or become compromised, several negative scenarios would be possible. For example, both users could report bad results that would be marked as correct if there was not enough replication (in practice, this number is not higher than three, so two users would build a quorum), or they could report a much higher computation time/value than they had to use in an attempt to obtain more credits. A possible solution for this problem would be a trust-based system, where peers would have a reputation based on their past actions. This was presented in [23], where weighted voting mechanisms were proposed, and clients were classified according to the results of the computation.

- **Exploiting Network Topology** — Another possible advantage of the BitTorrent protocol would be the possibility to take advantage of the network topology. Clients could give a higher priority to peers on the same Local Area Network, reducing the traffic generated to the outside. Bram Cohen's BitTorrent client has an option turned on by default, –use local discovery, that scans the local network for other clients with the desired content. Another possibility is using an approach similar to the one used in the Julia Content Distribution Network [5], in which nodes gather statistics about the network conditions as the download progresses, and then contact closer nodes (in terms of latency and bandwidth).
- **Integration with BOINC** — To allow for an easy integration with BOINC, the current prototype implementation has been completed in the same language as BOINC, C++. This minimized the conflicts and number of additional software packages needed. Additionally, if the initial "seed" is left running on the network, failure of any or all peer P2P data nodes to distribute data to a given client essentially causes a fallback to the standard centralized nature that BOINC currently implements, greatly reducing the risk of overarching data failures.

5.1 Proposed Scenario

In this new architecture a BitTorrent tracker is installed on the central server, and a port is defined to receive client requests (normally 6881). We decided to use a centralized tracker because the decentralized alternative is very recent, and the maintenance and construction of the DHT requires each peer to maintain an orthogonal set of neighbors within the DHT, and pay the communication costs of maintaining the DHT in the face of high rates of churn [14]. A .torrent file is created for every input file that should be downloaded through BitTorrent, pointing to the tracker in the central server: file.data -> file.data.torrent. The original file and its torrent counterpart are hosted on a project data server. To start sharing the file, the BOINC server must run a BitTorrent client to act as a seed and announce itself to the tracker. The .torrent file is related to the data file through the work unit. When creating work, a tag <bittorrent/> is added to the file info of the data file in the work unit template and the .torrent file itself is added as an input file.

Figure 1 shows the architecture and highlights the steps of a file transfer:

(1) The client contacts the scheduler and asks for work. The scheduler then replies with a given work unit and reference to a .torrent file that represents an input file made available via BitTorrent;
(2) The client then downloads the .torrent file through normal HTTP from the specified Data Server;
(3) After downloading the .torrent file, the BOINC client initiates the local BitTorrent client with the .torrent as an argument. The BitTorrent library then contacts the tracker defined on the file and receives list of peers;
(4) The client contacts the chosen peers and the BitTorrent protocol is used to download the subsequent file chunks and re-assemble the input file for processing by the local BOINC client.

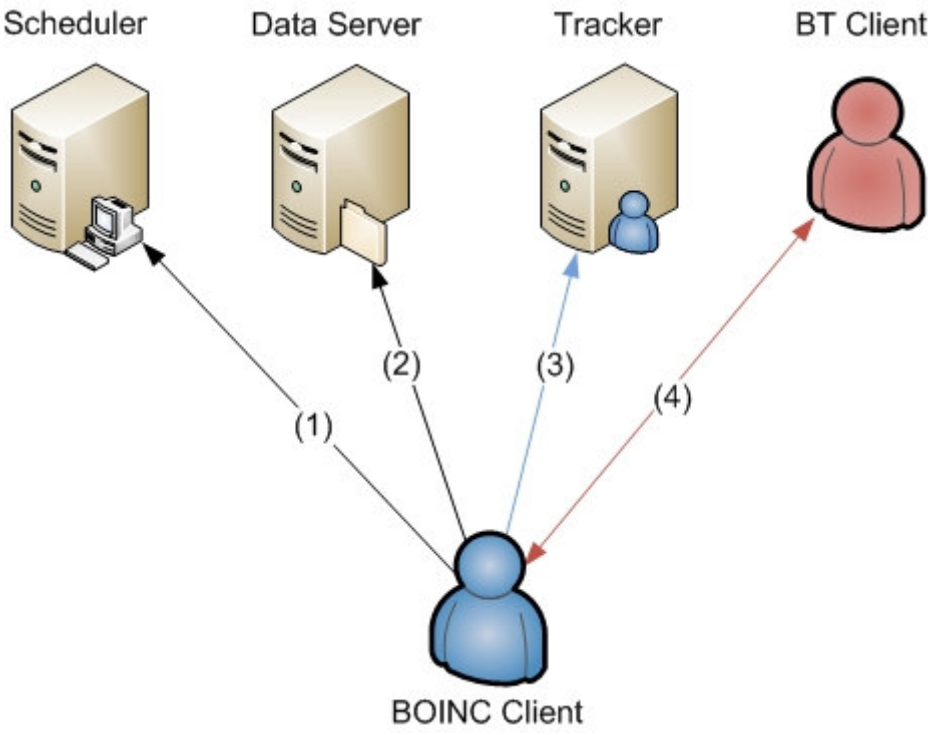

Figure 1 – BT BOINC file transfer

The downloaded input file is then checked for integrity through its hash and size. After being verified, it is used for the processing of its workunit. The rest of the process is unchanged from the original BOINC.

This architecture can help reduce the load on the server and possibly improve transfer times for projects where input files are large and shared by many work units. It can provide new opportunities for projects that were previously limited by bandwidth issues on their server and, by improving the data distribution, speeding up the scientific research behind the projects. On the other hand, this approach is likely to be received with skepticism, if not resistance, for two main reasons: (i) users are not willing to share their bandwidth when there is no direct benefit[§] and the alternative works; (ii) BitTorrent, like other P2P systems, is normally associated with piracy and illegal downloads, which taints its reputation; and, (iii) besides motivation, security can also be an issue since, to operate in good conditions, ports must be opened which increases users' vulnerabilities (not necessarily because of the BitTorrent protocol).

Recent experiments on the XtremWeb platform using BitTorrent showed promise [21], and should be and indicator of what to expect in this case. It is important to run experiments on a medium to large scale to ascertain the impact of the BitTorrent protocol on BOINC, and to determine the scenarios on which it will have the best performance. We expect to find a crossover point in performance in terms of file size and number of nodes sharing the file between the original BOINC and this version.

6 Approach 2: Super-Peers and Secure Data Centers

BitTorrent can fairly effectively solve the data needs of BOINC as they relate strictly to distribution. However, it has limited security beyond ensuring file integrity and has no notion of grouping or peer hierarchy. For volunteer computing communities, security can be a much larger issue than simply guaranteeing data validity. Due to the sensitive and vulnerable nature of Desktop Grids, and in particular volunteer networks used for research purposes, whose user community is volatile, it is critical not only for data integrity and reliability to be ensured, but also that peer nodes are secure from malicious attacks.

[§] unless network utilization is added as a contributing factor to the credit ratings that users are given for participating in the network, this would, however, require adaptation and monitoring of BitTorrent file transfers

This requires a number of steps, and can be implemented in a variety of fashions, each with their own benefits and tradeoffs. The easiest, and perhaps most susceptible to attacks is a pure P2P network, in which any node is allowed to receive and share information with any other node on the network, as BitTorrent does. Although this is perhaps the most efficient use of a P2P network, and could potentially reap the largest rewards as far as potential disk space capacity and network bandwidth utilization, it is also the most dangerous, given its requirements for opening ports and generalized policy that all nodes participate on an equal level. Since any node in this scenario has the capability to flood the network with false information, regardless of whether it is later discarded as invalid, the probability that this will happen is much greater than in a restricted network, where only "trusted" peers are allowed to act as data providers and message relaying, or rendezvous, nodes.

Secure data centers are a way of implementing a super-peer topology for data sharing that would restrict the set of peers that are allowed allowed to propagate data. In this scenario, policies could be set by each BOINC project as to which participants, if any, are allowed to host and redistribute data. Beyond simply restricting data center membership, policies could also be introduced to govern the relative sensitivity of data and retention policies. Adding these new types of functionality would allow for more advanced scenarios, although with the additional costs of software and network complexity.

The Secure Data Center ideas discussed here are currently in the process of being implemented in the form of a software middleware entitled "Peer-to-Peer Architecture for Data-Intensive Cycle Sharing" (P2P-ADICS) [17], which was briefly introduced in section 2. P2P-ADICS is building a super-peer architecture for data sharing that focuses on allowing for the dynamic configuration of group membership that facilitates creating secured data-caching overlay networks that coexist with the conventional super-peer discovery overlay for bootstrapping purposes. In this scenario, to implement the data sharing aspects of BOINC, a new overlay network would be created which contains only those nodes that have been promoted to datacenters, within this overlay, data centers propagate data amongst themselves and serve requests to the underlying worker layer.

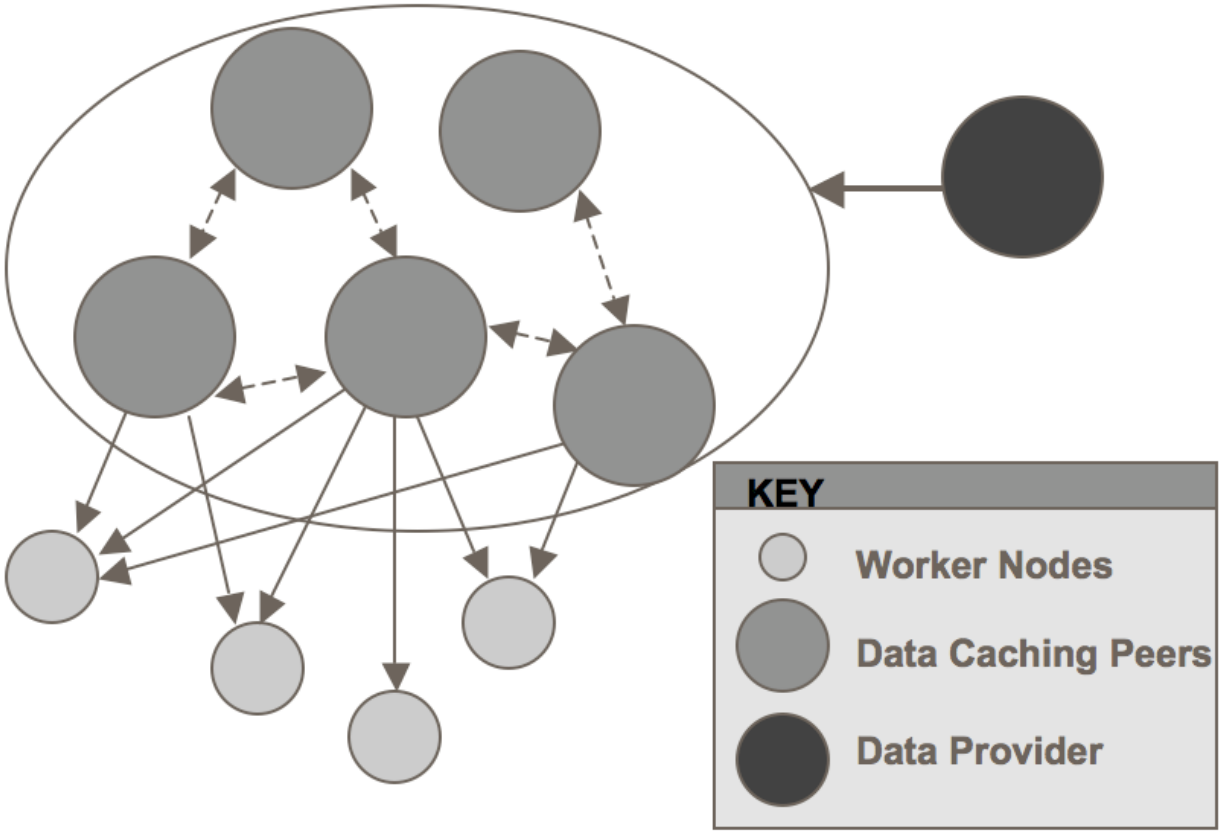

Figure 2: Representation of overlay: data provider, caching nodes and worker nodes

Figure 2 gives a visual representation of how the different components in this network relate to one-another, after the initial discovery process has taken place. In this discovery phase (not pictured), a worker node sends a request to known access points on the data center overlay, which responds with an updated list of data centers that the worker

node can use to harvest data. Failing to discovery anyone, the worker node will directly contact the data provider to request a data center reference.

In the following, the four target issues identified earlier in §4 are discussed, with a brief overview of how they relate to Secure Data Center integration, and the preliminary implementation states of P2P-ADICS.

- **Firewall & Router Configuration** — Depending on an individual projects configuration, firewall and router issues could be a potential problem, or a complete non-issue. In a free-for-all system where any member node was permitted to be a data center, there could obviously be problems with that node being behind aNAT, and the tradeoff between "punching holes" in the firewall and the potential benefit of the node's available network bandwidth would have to be determined. For more restricted systems, in which pre-specified static or semi-dynamic[**] nodes are dynamically promoted to be data centers as the network requires, firewall and router issues could be minimized, for example, through enforcing eligibility criteria for data centers to only those nodes that have a publicly addressable network space.
  Current design of P2P-ADICS is working with the assumption that a more secured sharing will be desired and enforced, which requires data center peers to be publicly accessible machines, thereby for the moment forgoing the potential pitfalls of attempting to implement automatic firewall configuration, leaving this as a future implementation issue.
- **Malicious Users** — As with Firewall & Router Configuration, the issue of how much relative freedom network participants have to manipulate the network will depend on the individual policies of each hosting project. In the most restrictive case, the only nodes that would be allowed to propagate data would be well known and trusted, thereby affording the same level of security currently available in the centralized network. In looser security configurations, which are configured to harvest more participant network resources, the security issueswould be roughly equivalent to BitTorrent as discussed in §5. The advantage of the system proposed here is there are middle-ground options that lying between these two extreme alternatives that could be exploited.
  P2P-ADICS relies on the data signing and validation procedures currently utilized by BOINC, which essentially guarantee that requested data will be what is ultimately retrieved. However, to effectively distribute a single data file from multiple data centers to an individual host, BitTorrent-style file-swarming techniques are being investigated, which require two-level hashing of data, once on the individual chunks, and once on the entire file. Therefore, this additional chunk-level hashing is in the process of being implemented in an attempt to prevent malicious users from propagating "bad chunks," to the network.
- **Exploiting Network Topology** — Similar to the mechanisms employed by BitTorrent [§5] and the Julia Content Distribution Network [5], network proximity would have to be determined to adequately map nodes and decide if any are on a local network. However, if the network parameters are set to limit the participants to known hosts, then the likelihood of internal LAN nodes being available to a given peer as a data center is significantly diminished. In these cases, a two-tier system of data servers is envisioned: one, in the traditional case, which meets certain selection criteria, but is available on the larger network via a public address; and another which has also met the selection criteria for a "trusted node" yet is unavailable to the larger network, however is available to distribute files to local peers. Alternatively, LAN data centers could have lower security requirements placed upon them, as the data is digital signed to verify integrity, however, this could allow for malicious exploits involving the reporting of false results should multiple recipients on the same LAN be given identical tasks to compute.
  P2P-ADICS, through its underlying reliance on P2PS currently use sUDP multicasting for LAN discovery of data servers, and KaZaA-style "known peers" forWAN discovery. As the project progresses, technologies such as those employed by the Julia Content Distribution Network will be explored for more advanced network topology exploitation.
- **Integration with BOINC** — The Secure Data Center approaches outlined here, in the form of P2P-ADICS, would demand more radical changes and a larger software stack than the BitTorrent implementation proposed in §5. This is primarily due to two distinct areas: internal integration with BOINC and external library dependencies. Regarding internal integration, the BitTorrent solutions are fairly straightforward, the centralized HTTP server contact address is simply replaced with the corresponding tracker. In the off chance case in which no peers are mirroring the data, the client simply downloads from the centralized

[**]In this instance, semi-dynamic, is referring to nodes that have gone through some pre-screening that verifies them as good candidates for data-centers, such as obtaining a specific certificate or accumulated substantial project credits. However, when they actually perform as data centers is determined dynamically based upon network properties.

server, as it would have under the current implementation. For P2P-ADICS, to ensure a comparable level of certainty, an if/else statement would have to be injected into the client code, whereby if a lookup on the P2P network failed, clients could resort to traditional download means. Although this solution adequately manages the problem, it could incur severe latencies if not implemented correctly.
Regarding external library dependencies, BitTorrent solutions could only require the addition of a single C++ BitTorrent libary, which could be used to broker BitTorrent downloads. P2P-ADICS is currently being built atop P2PS, which is implemented in Java. This creates a client dependency on a JRE. There are two possible solutions to this problem: (i) add a JRE to the required software to run BOINC, which could potentially limit adoption of P2P-ADICS; and, (ii) create a light-weight client-side C++ implementation of the P2P-ADICS client download capabilities, thereby limiting the JRE requirement to nodes that wish to operate as data centers. The current design and plans for P2P-ADICS is pursuing option i in an attempt to build a working system, and will later reassess the necessity of option ii based upon feedback from the BOINC user-community.

In [8] a more general cycle-sharing paradigm utilizing Peer-to-Peer systems to distribute work units was presented[††]. Although the work presented there is more generalized, the fundamental "dynamic caching" and data distribution aspects are coherent with the ones presented here, and the results and arguments therein can be directly applied to the scenario proposed here. Specifically, [8] presents an argument that using dynamic data caching, whilst knowing the network and data properties, can allow for a more efficient configuration of data server replication, as opposed to the current static sized set used by BOINC projects.

Based upon the preliminary results of [8] and the arguments presented here, it is our belief that decentralized data centers can prove to be both valid and useful solutions to distributing data in Desktop Grid environments. There is, however, a tradeoff between functionality and complexity that needs to be adequately addressed and balanced if such technologies are to be adopted by production environments such as BOINC. P2P-ADICS is an ongoing research project attempting to build a system that can address the needs of scientific users, while maintaining the benefits of a decentralized network that utilizes available network properties at much as possible.

7 Conclusions

In this paper we have argued that the current centralized client/server architecture applied by BOINC and other Desktop Grid systems for data distribution is limiting and costly, and these projects would benefit from P2P data distribution technologies. Specifically, we have presented two approaches for large-scale data management in Desktop Grid domains: one, based directly upon the BitTorrent protocol, and another employing a decentralized unstructured P2P network. For both of these potential solutions, we have provided the reader with arguments for and against, weighing the relative costs and benefits of uptake, as well as giving the current status and directions we are undertaking in our work in these areas. It is hoped that the ideas presented here will promote the discussion of Peer-to-Peer data distribution not only in the BOINC and Desktop Grid groups, but also to the wider scientific community, encouraging others to explore P2P as a valid and useful approach for data distribution.

Acknowledgements

The authors wish to thank Pasquale Cozza and Domenico Talia of DEIS University of Calabria, and Carlo Mastroianni at ICAR-CNR for their contributions and help. This work was supported by the CoreGRID Network of Excellence, the Center for Computation & Technology at Louisiana State University, and EPSRC grant EP/C006291/1.

[††]Authors Kelley and Taylor are in the overall design of this system, which is led by ICAR-CNR.

References

[1] David Anderson. BOINC: A System for Public-Resource Computing and Storage. In Proceedings of the 5th IEEE/ACMInternationalWorkshop on Grid Computing, Pittsburgh, USA, November 2004.

[2] David Anderson. Volunteer Computing: Planting the Flag. PCGrid 2007Workshop, Long Beach, March 30 2007.

[3] H. Balakrishnan, F. Dabek, M.F. Kaashoek, D.R. Karger, D. Liben-Nowell, R. Morris, and I. Stoica. Chord: a scalable peer-to-peer lookup protocol for Internet applications. Networking, IEEE/ACM Transactions on, 11.

[4] Berkeley Open Infrastructure for Network Computing (BOINC). See web site at: http://boinc.berkeley.edu/.

[5] Danny Bickson and Dahlia Malkhi. The Julia Content Distribution Network. 2nd Usenix Workshop on Real, Large Distributed Systems (WORLDS '05), San Francisco, USA, December 2005.

[6] F. Cappello, S.Djilali, G.Fedak, T.Herault. F.Magniette, V.Neri, 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.

[7] Bram Cohen. Incentives build robustness in BitTorrent. Proceedings of IPTPS, 2003

[8] Pasquale Cozza, Ian Kelley, Carlo Mastroianni, Domenico Talia, and Ian Taylor. Cache-Enabled Super-Peer Overlays for Multiple Job Submission on Grids. To be published ISC 2007 CoreGrid Workshop, 2007.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 29–43, New York, NY, USA, 2003. ACM Press.

[10] Gnutella Project. See web site at: http://www.gnutella.com/ .

[11] 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.

[12] P. Kacsuk, N. Podhorszki, and T. Kiss. Scalable Desktop Grid System. CoreGRID Technical Report TR-0006, MTA SZTAKI, University of Westminster, 2005.

[13] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao Oceanstore: An architecture for global-scale persistent storage. In the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[14] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. IEEE Conference on Computer Communications (INFOCOM), 2005.

[15] J. Liang, R. Kumar, K. W. Ross. The KaZaA Overlay: A Measurement Study. Computer Networks Journal, Oct. 2005.

[16] M. Litzkow, M.Luvby, and M.Mutka. Condor - A Hunter of IdleWorkstations. Pages 104-111. 8th International Conference on Distributed Computing Systems (ICDCS).Washington, DC, 1988.

[17] Peer-to-Peer Architecture for Data-Intensive Cycle Sharing (P2P-ADICS). See web site at: http://www.p2p-adics.org/.

[18] Sudharshan S. Vazhkudai, Xiaosong Ma, VincentW. Freeh, JonathanW. Strickland, Nandan Tammineedi, Stephen L. Scott. FreeLoader: Scavenging Desktop Storage Resources for Scientific Data. sc, p. 56,Ê ACM/IEEE SC 2005 Conference (SC'05),Ê 2005.

[19] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for Peer-to-Peer Distribution Computing in a Heterogeneous, Decentralized Environment. Proceedings of the Third International Workshop on Grid Computing, 2002.

[20] Ian Wang. P2PS (Peer-to-Peer Simplified). In Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, pages 54–59. Louisiana State University, February 2005.

[21] BaohuaWei, G. Fedak, and F. Cappello. Scheduling independent tasks sharing large data distributed with BitTorrent. pages 219-226. Grid Computing, 2005. The 6thIEEE/ACM International Workshop on, IEEE ComputerSociety, 2005.

[22] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. UCB Tech Report UCB/CSD-01-1141, University of California, Berkeley, 2001.

[23] Gheorghe Cosmin Silaghi, Louis M. Silva, Patricio Domingues, and Alvaro E. Arenas. Tackling the Collusion Threat in P2P-Enhanced Internet Desktop Grids. Presented in a CoreGRID Workshop, Crete,Greece, June 2007.

[24] Chandramohan A. Thekkath , Timothy Mann , Edward K. Lee. Frangipani: a scalable distributed file system. pages 224-237. Proceedings of the sixteenth ACM symposium on Operating systems principles, October 05-08, 1997, Saint Malo, France