Distributed Data Storage for Opportunistic Grids*

Raphael Y. de Camargo
Dept. of Computer Science
Universidade de São Paulo, Brazil
rcamargo@ime.usp.br

Fabio Kon (advisor)
Dept. of Computer Science
Universidade de São Paulo, Brazil
kon@ime.usp.br

ABSTRACT

Grid applications typically need to deal with large amounts of data. The traditional approach for data storage is to employ high-performance dedicated servers with data replication. However, a class of computational grids, called opportunistic grids, focus on the usage of idle resources from shared machines. These machines normally have large quantities of unused storage space that could be used when the machines are idle, allowing opportunistic grids to share not only computational cycles, but also storage space.

In this work, we present the initial design of OppStore, a middleware that provides reliable storage using the free storage space from shared grid machines. The storage can be transparently accessed from any grid machine, allowing easy data sharing among grid users and applications. The system uses a two-level peer-to-peer organization to connect grid machines in a scalable and fault-tolerant way. To deal with resource heterogeneity, we developed the concept of virtual ids, which allows the creation of virtual spaces located on top of the peer-to-peer routing substrate. These virtual spaces enables the middleware to perform heterogeneity-aware, load-balancing selection of storage sites using multiple simultaneous metrics.

Categories and Subject Descriptors

C.2 [**Computer-communication Networks**]: [Distributed systems]

General Terms

Design, Performance, Reliability

1. INTRODUCTION

In addition to powerful CPUs, applications executing on computational grids [2,8,11] typically require an infrastructure for manipulating large amounts of data. Stored data may be shared by several applications, as output data from an application may be used as input for several other applications. Consequently, a grid infrastructure needs a reliable, high-capacity storage system that can be easily accessed from any point in the grid.

Traditionally, data storage in computational grids uses dedicated storage servers, with data replicated across several servers and managed by replica management systems [3,4,20]. These systems usually target high-performance computing platforms, with application that requires very large amounts (terabytes or petabytes) of data and run on supercomputers connected by specialized high-speed networks. But this infrastructure is only available to institutions that can afford the high costs associated with it.

Opportunistic grids are a class of computational grids focusing on the usage of idle computing resources [5, 14, 17]. These resources include several types of machines, such as workstations, PCs, and computing clusters. The motivation for "opportunistic" is that these machines remain idle for the vast majority of time. The main shared resources are the idle processor cycles of the machines. But these shared machines often also have large amounts of unused storage space in their local disks. Combining the unused space of a few hundred machines, we can easily achieve several terabytes of storage space. Moreover, using these machines for storing data would improve resource utilization and minimize the need to purchase extra hardware to store application data. This enables a low-cost solution for data storage on institutions that have limited budget, for example in developing countries.

But using shared machines for data storage is a difficult task. Machine owners sharing their resources with the grid should have their Quality of Service (QoS) preserved, implying that machine resources should only be used during idle periods. Also, shared machines are frequently turned off, for example at night or during weekends. A distributed storage system using these machines must ensure data availability in this highly dynamic and unstable environment. Finally, computational grids may encompass tens of thousands machines, requiring the system to be self-organizing and highly scalable.

A similar scenario, in which available free space of several desktop machines is used to store and distribute content, is explored by peer-to-peer file systems, such as CFS [6] and PAST [21]. Data location in these systems is implemented by a distributed hash table (DHT). But an important problem with most DHTs is the poor load balancing guarantees they offer. Issuing random ids to nodes results in some nodes being responsible for id ranges $O(\log n)$ times larger than other nodes. Moreover, the heterogeneity of nodes is not considered. To solve these problems, load balancing algorithms for DHTs have been proposed recently [6, 13, 15, 23], with most of them using the concept of virtual servers. Using virtual servers, the amount of the id space allocated to each server can be dynamically moved from one server to another. But this approach

*This work is supported by a grant from CNPq, Brazil, process #141966/03-3.

MDS '06, November 27-December 1, 2006 Melbourne, Australia

has the drawback of requiring large amounts of maintenance information.

In this work, we propose *OppStore*, a middleware that explores the usage of a peer-to-peer overlay network for storage of data in non-dedicated machines in the context of opportunistic grids. The middleware organizes the grid machines in a federation of clusters, organization used by several existing opportunistic grids, such as Condor-G [12], InteGrade [14], and OurGrid [5]. At the cluster level, a cluster manager is responsible for several data repositories, running on the cluster shared machines. At the grid level, cluster managers are connected by a structured peer-to-peer overlay network, forming a global federation of data storage clusters. To improve data availability in face of the dynamism of a grid environment, OppStore uses an information dispersal algorithm [7, 18] to code a file into several redundant fragments. The system then scatters these fragments into several grid clusters chosen randomly.

Peer-to-peer overlay routing is performed using a distributed hash table (DHT) based on Pastry [22]. To deal with node heterogeneity, we introduce the concept of *virtual ids*. Using virtual ids, in addition to the original node id provided by Pastry, each node receives an extra virtual id, creating a virtual id space located on top of the Pastry id space. These virtual ids can change dynamically as nodes join and leave the system, and when the load on nodes change. The amount of virtual id space allocated to each node is determined by the node capacity, allowing dynamic load-balancing among heterogeneous nodes. In the OppStore scenario, the objective is to maximize data availability and to place data on nodes with abundant storage space. Consequently, we define as the capacity of machines the product between its available space and mean availability[1]. Although in OppStore we initially use a single virtual space, it is possible to instantiate multiple simultaneous virtual spaces, allowing load-balancing according to different metrics.

We expect the following contributions from this work:

- Design of OppStore, a distributed storage middleware for opportunistic grids that uses non-dedicated machines for the storage of user and application data in a feasible, scalable, and fault-tolerant way. The system uses a two-level peer-to-peer organization to connect grid machines;
- The novel concept of virtual ids, which enables load-balancing in DHT-based peer-to-peer networks composed of heterogeneous nodes. It allows the creation of multiple virtual spaces located on top of the original DHT id space.

2. RELATED WORK

This work intersects with three research areas, namely grid data management, distributed storage systems, and load-balancing in peer-to-peer systems.

2.1 Grid Data Management

FreeLoader [25] aims to use free desktop storage space and I/O bandwidth to store scientific data. The system divides a file into several fragments to improve performance. Similarly to our work, it targets non-dedicated resources for data storage. But it only considers static sets of machines from a single cluster, while we deal with dynamic sets of machines distributed across several clusters. Also, Freeloader does not consider load-balancing and machine availability when choosing storage sites.

[1]In this work, we define machine availability as the fraction of the total time in which the machine is available for data download and upload.

JuxMem [1] implements a data sharing service for grid applications by joining the concepts of peer-to-peer and distributed shared memory. As in our work, it organizes grid machines as a federation of clusters connected by a peer-to-peer model, with a node elected as cluster manager in each cluster. Differently from our work, the authors focus on the development of a writable distributed shared memory for grid applications. This requires maintaining several full replicas of stored data, which incurs large storage and network overheads, specially when operating with non-dedicated machines, where the replication level must be higher. Also, they do not employs mechanisms for load-balancing and the availability characteristics of machines is not considered when choosing storage sites.

A common technique for data grids is the usage of data replication in conjunction with a replica location system [3, 4, 20]. Some replica management systems [4,20] use compression schemes, such as Bloom filters, allowing replica managers to have complete knowledge about replica locations on the grid. The search mechanism is fast and the system has a high degree of fault-tolerance. But due to global knowledge of replica locations, these systems have limited scalability, with possibly expensive table updates. Also, the servers maintaining the replica locations are usually configured statically.

Cai *et al.* [3] built a replica location system using the Chord [24] distributed hash table. The objective was to provide self-organization and improved fault-tolerance and scalability for the replica location system. The system only deals with replica location, while OppStore also deals with storage in non-dedicated repositories, including the selection of appropriate repositories. Also, the proposed replica location service employ load balancing only for replica location queries, and considers that all the servers have equal capacities. Our system uses a load-balancing technique that considers the heterogeneity of shared machines in grid clusters.

2.2 Distributed Peer-to-Peer Storage Systems

PAST [21] is a peer-to-peer, read-only storage system built over Pastry [22]. The system connects machines through a Pastry DHT, and uses this infrastructure to route files for storage. The system stores full files on the machines, using replication to provide fault-tolerance. OppStore supports both file replication and erasure coding. More importantly, there is little support for load distribution in PAST, with support only for storage request forwarding. Also, the system does not consider machine heterogeneity. In our system, we use an heterogeneity-aware load-balancing algorithm that operates directly in the routing algorithm.

The Cooperative File System (CFS) [6] also provides a read-only peer-to-peer storage system, but uses the Chord [24] DHT algorithm. CFS breaks files into several blocks and stores multiple replicas of each block. A CFS client is responsible for the conversion from block to file system semantics. CFS uses virtual servers for load balancing, allowing machines to instantiate a variable number of virtual servers. The drawback of this approach is that, when virtual servers are instantiated or relinquished, a large amount of data copying may be necessary. Also, the usage of virtual servers requires higher bandwidth overhead to maintain the overlay network. Our storage system prevents both problems by employing an alternative loading balancing technique based on the concept of virtual ids and by the usage of a two-level scheme for file storage.

OceanStore [16] creates a global-scale persistent storage for mutable data. It uses Plaxton trees and Bloom filters to perform data location and data can be stored using replication or erasure coding. To allow data updates in the presence of file replicas, Byzantine agreements protocols are used. Finally, introspection mechanisms are employed to cluster related files and for replica management. All these features of the system come at the cost of high implemen-

tation complexity. Also, the system does not consider node heterogeneity and does not employ load-balancing algorithms when selecting data storage sites.

2.3 Load Balancing in DHTs

Most of the works on load-balancing of DHT tables use the concept of virtual servers [6, 13, 15, 19, 24]. The most common approach is to let each node to instantiate up to $O(\log n)$ virtual servers, where n is the number of nodes [6, 24]. While straightforward to implement, this approach has some important drawbacks, such as the large amount of maintenance information, which occurs because each node needs to keep information about $O(\log n)$ virtual servers. Karger and Ruhl [15] proposed a variation of the virtual servers approach by allowing each server to assume the identity of exactly one of $\log n$ virtual servers. It has the advantage of balancing the load without the need of tracking many virtual servers, but has the drawback of providing less control when adjusting load distribution. An important drawback of using virtual servers is that the process of destroying and instantiating new virtual servers every time that load needs to be reallocated is expensive, as it requires the construction and update of several tables. Also, using virtual servers, the application is limited to a single routing space[2]. In contrast, with the virtual ids we propose, changes in id ranges are cheaper to implement and multiple simultaneous id spaces can coexist with a small overhead.

The concept of virtual servers is also applied to dynamic load balancing, where virtual servers from an overloaded node are transferred to an underloaded one. These load-balancing schemes normally use the concept of directories to store load-information [13, 19]. Nodes send periodical updates about their load to one of several directories, which control exchanges of virtual servers between nodes. Using directories incurs the overhead of requiring nodes to send periodic updates to directories. Using virtual ids, load is exchanged directly between neighbors, what is incompatible with the use of directories. On the other hand, with virtual ids it is possible to fine tune load exchange between nodes and load can be balanced according to multiple metrics.

Using a different approach, Karger and Ruhl [15] propose that a server may exchange part of its id space with its neighbors to perform load-balancing. This approach is similar to ours, but in their work exchanges are performed only between immediate neighbors. In a recent work, Xu and Bhuyan [26] proposed an algorithm which uses file access history and node heterogeneity to perform load balancing. Similarly to our work, a node divides its id space with its neighbors. But in both works, the node real id is changed instead of a virtual id. Changing real ids is more costly than changing virtual ids, since it is usually implemented by a node departure operation followed by a node joining one. More importantly, their load balancing scheme permits a single virtual space. In a sense, we could say that virtual ids could be used to generalize these schemes to allow other metrics to be used, such as bandwidth and node churn history.

An important factor to consider is that, depending on the application, changes in the id range allocated to a node may require data transfer between nodes. This typically occurs in data storage applications, which use the same id to store and to locate a file in the peer-to-peer network. The analysis of the amount of data copying is dependent on the application and can involve the replication strategy used. But, when using virtual ids, the possibility of having more than one space for message routing provides more flexibility to the application. For example, OppStore can dynamically balance the amount of data uploaded to a node without requiring any data transfer during virtual id range reallocation.

[2]Actually, more than one routing space can be obtained by simultaneously maintaining multiple overlay networks. But this procedure would be too expensive.

3. MIDDLEWARE DESIGN

We designed a middleware to permit reliable and efficient storage of data using the free storage space from idle machines in opportunistic grids organized as a federation of clusters. The middleware uses a peer-to-peer substrate to route data storage and retrieval requests to the target machines. In the following section, we briefly describe the OppStore architecture and outline its main protocol. We then analyze the usage of virtual ids to deal with machine and cluster heterogeneity.

3.1 Middleware architecture

OppStore is structured as a federation of clusters, with each cluster containing a Cluster Data Repository Manager (CDRM), which manages several Autonomous Data Repositories (ADR), one on each cluster node. Consequently, the system can be divided in two-levels, the grid level and the cluster level. CDRMs form a structured overlay network, using the Pastry [22] distributed hash table (DHT) algorithm. We use 160 bit ids to uniquely identify CDRMs and stored data. Each cluster is responsible for storing data from a portion of the DHT id space. ADRs are simple data repositories that accept requests for data storage and retrieval in a single machine. They use very few system resources, and can be configured by the machine owner, for example, to allow data upload and download only when the machine is idle or at any time, but limiting the borrowed bandwidth. Figure 1 shows the main OppStore components.

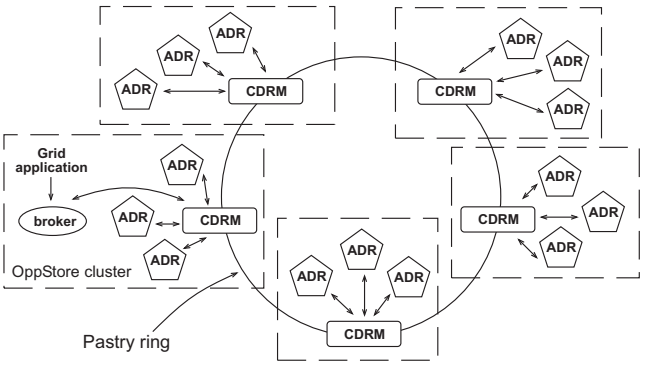

Figure 1: OppStore Architecture.

This two-level architectural design facilitates the management of system dynamism. If the system organized all grid machines in a single peer-to-peer overlay network, the constant changes in machine availability, which typically occur in opportunistic environments, would have to be treated as node joining and departure operations, which are expensive operations. When using a two-level design, this dynamism can be managed in the local cluster, by the cluster manager.

The federation structure also allows the system to disperse grid data throughout the Grid. During storage, the system slices the data into several redundant coded fragments and stores them in different grid clusters. This distribution improves data availability and fault-tolerance, since fragments are located in geographically dispersed clusters. When performing data retrieval, applications can simultaneously download file fragments from the closest clusters, enabling efficient data retrieval.

3.2 Data storage and retrieval

Clients access the distributed storage system through a *broker*, which is responsible for contacting other OppStore components to perform file storage and retrieval operations. Several types of data can be stored in a grid system, with each type having different requirements. OppStore allows a client application to choose one of two storage modes: perennial and ephemeral.

The *perennial mode* is used for data with long lifetimes. The broker performs file storage in two phases. In the first phase, the broker breaks the file contents into several redundant fragments using an information dispersal algorithm [7, 18] and evaluates their secure hashes. The broker then sends the list of fragment hashes to the cluster CDRM, which routes messages in the overlay network to other CDRMs using the fragment hashes as message ids, requesting the addresses of suitable ADRs to store the fragments. The cluster CDRM then sends the ADR address list back to the broker, which uploads the fragments directly to the ADRs. In the second phase, the broker constructs a *File Fragment Index (FFI)*, containing the location and the secure hash of each fragment. The broker then set the *FFI id* as the secure hash from the fragments hashes and requests the FFI storage to the cluster CDRM.

Retrieving a file is similar. The broker retrieves the FFI of the file and downloads its fragments directly from the ADRs. The broker then checks the fragments integrity and reconstruct the file. Consequently, we can say that data stored in OppStore is self-verifying.

We should emphasize that file contents are not routed through the overlay network, only the fragment ids and the FFI. File contents are transferred directly from the broker to the ADRs and vice versa. This is important because individual files may have sizes of several gigabytes. If they had to be routed through several hops in the overlay network, this would overload the CDRMs.

Also, placing fragment storage locations in the FFI provides an important advantage: the fragment locations are not tied to the cluster responsible for their ids. In other words, when the id range for which a CDRM is responsible changes, there is no need to move the fragments between clusters. The FFIs are still retrieved using their ids and, consequently, would need to be moved, but their size is small compared to file contents. And as we will see in Section 4, in some cases even the FFIs will not need to be moved.

The *ephemeral mode* is used for data that requires high bandwidth and only needs to be available for a few minutes. This class of storage would be used to store checkpointing [9, 10] data and temporary application data. An example of temporary data occurs in workflow applications, where data output by one application stage is used by a later application stage running in the same cluster. In this storage mode, the system stores the data only in the local cluster and uses data replication to provide fault-tolerance.

4. VIRTUAL IDS

OppStore determines the storage location of fragments by routing the fragment identifier to the CDRM responsible for that identifier. But this scheme does not take cluster heterogeneity into consideration. To allow OppStore to consider this heterogeneity when selecting storage sites, we included the concept of *virtual ids* in our routing algorithms. In addition to its original id, each node (CDRM in OppStore) can receive additional virtual ids. Each virtual id is associated to an extra virtual id space located on top of the original id space of the DHT. This virtual space is much cheaper to maintain than the original one and virtual ids can be changed with little overhead. The low maintenance cost allows simultaneous operation of multiple virtual spaces.

In this section, we outline how we extended the Pastry [22] algorithm to include virtual ids. Although we use Pastry as example, the virtual ids concept can also be implemented in other peer-to-peer substrates. In the following section we provide a very brief overview of Pastry[3] and then describe the main virtual id protocols for a single virtual space. The extension to multiple virtual spaces is straightforward, only requiring the execution of the same protocols on the other virtual spaces.

4.1 Pastry overview

Pastry provides a peer-to-peer routing substrate that implements a distributed hash table. Each Pastry node receives a random node id from the id space. A message routed through Pastry is guaranteed to reach the node with the closest id to the message id. To perform routing, ids in Pastry are converted to a number in base 2^d, where d is a Pastry parameter. At each step, a message is routed to a node whose id is numerically closer to message id. Pastry nodes maintain a *leafset* table, containing the node ids of the l numerically closer nodes. Before forwarding a message, a Pastry node checks whether the target node is in its leafset and, if positive, delivers the message to that node.

4.2 Virtual ids overview

When using virtual ids, nodes are organized in a virtual id space over the underlying peer-to-peer routing infrastructure. In this space, the virtual ids can be changed to reflect the heterogeneity of nodes and to adapt to dynamic environments where nodes constantly leave and join the network or change their characteristics. Each node in the virtual space receives responsibility for a virtual id range proportional to its capacity. This capacity can be a function of one or more metrics, such as node availability, storage capacity, bandwidth, and processing power.

When a node connects to the network, the Pastry joining protocol is executed, with the joining node receiving a random Pastry id. The joining node then starts the *virtual space partition* protocol, contacting the nodes from its Pastry leafset to obtain their virtual ids and capacities. Based on this information, the joining node partitions the virtual id space range covered by these nodes, giving to each neighbor a range proportional to its capacity. The joining node then sends the new virtual ids to these nodes.

The virtual space partition protocol is also executed when a node leaves the network or changes its capacity. It guarantees that the virtual id space from a subset of the nodes will always be partitioned according to their capacities. Using this protocol the system can easily adapt to highly dynamic environments.

Each node maintains a table called *virtual neighborset* containing the virtuals ids of its neighbors. The table is populated with information obtained during the virtual space partition protocol and is later used to improve the robustness of the virtual ids routing protocol described in the next section.

4.3 Routing in virtual spaces

To allow routing in the virtual id space, each node maintains an additional table, called *virtual leafset*. This table contains a list of successive nodes from the virtual id space, centered in the node original id. The virtual leafset maps the original id space covered by the Pastry leafset of a node into the virtual id space and allows the transition from the original id space to the virtual space. Figure 2 shows the relation between the original and virtual id spaces, the Pastry leafset, the virtual leafset, and the virtual neighborset.

The virtual routing algorithm allows routing a message from any node S to the node T responsible for the *message id* in the virtual id space. The idea is that since the virtual and original id spaces

[3]For more details of Pastry, refer to the original paper [22].

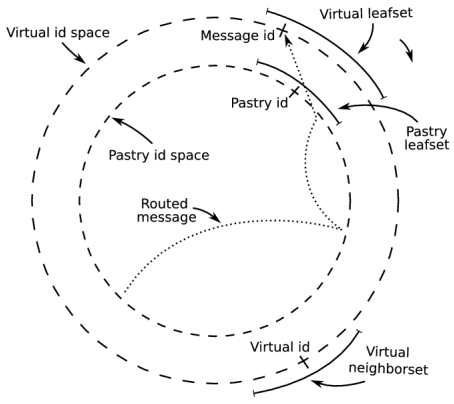

Figure 2: Virtual id space.

are superposed, all the routing can be performed in the original id space using the Pastry algorithm. The transition to the virtual space occurs when the message reaches a node whose virtual leafset contains the node responsible for the message id in the virtual id space. This transition is guaranteed to occur as long as the virtual leafset of a node covers at least the same id range as its pastry leafset.

The important consequence of using the underlying routing infrastructure to perform routing in a virtual space is that its maintenance is very cheap. The underlying overlay keeps all the costs of maintaining the network integrity and keeping the routing tables updated. This allows the simultaneous creation of several virtual spaces with low overhead. Moreover, the original id space is maintained and can also be used for message routing.

The virtual leafset needs to be updated when virtual ids of its members change. When a node finishes a virtual space partition protocol, it sends a message to the nodes with references to the changed virtual ids with the new values.

We emphasize that the virtual leafset and neighborset have a mean size similar to the Pastry leafset. Also their maintenance is very cheap, since they only need to be updated when some virtual id changes. Even in this case, the number of nodes contacted is also constant, and is typically in the order of 4 times the leafset size. Consequently, in contrast to using virtual servers, that requires a large increase in the overlay maintenance costs, using virtual ids we can augment a peer-to-peer substrate to deal with heterogeneity using very few extra resources.

4.4 Using virtual ids in OppStore

In OppStore, we define the capacity of each machine as the product of its mean availability and free storage space, and use as cluster capacity the sum of the capacities of all cluster machines. OppStore creates a single virtual id space, using the cluster capacities to define the CDRMs virtual ids. This virtual space is used for routing when selecting the cluster that will host a file fragment. The objective is to store fragments in clusters containing machines with higher availability and free storage space.

Routing and selection of CDRMs responsible for FFI ids is performed in the *Pastry* id space. Pastry ids are independent from the virtual ids and, consequently, CDRMs can change their virtual ids without requiring the migration of FFIs. This migration will only be necessary when CDRMs join or leave the Grid, which should be infrequent. Moreover, since fragment locations are stored in the FFIs, the fragment contents also do not need to migrate due to virtual and pastry id changes. Actually, tranfers of fragment contents is only necessary when a significant part of fragments from a file are lost due to ADR departures. In this case, the lost fragments are reconstructed. Consequently, using virtual ids, OppStore can adaptat to dynamic conditions with very low overhead.

5. RESEARCH METHODOLOGY

We will validate OppStore using two methodologies: simulations and experiments in a controlled real grid environment. The objective of the simulations is to determine how data is distributed in a grid composed of heterogeneous clusters. For example, we want to determine in how much the file retrieval success rate is improved when we use capacities based on the machine availability. In another simulation we intend to utilize a network topology simulator to evaluate the geographical distribution of stored fragments and determine good caching policies. Finally, we are using simulations to analyze the virtual ids protocols, for example to determine the amount of data traffic generated by the protocols.

To perform experiments in a real grid environment, we are integrating OppStore with the InteGrade opportunistic grid middleware [14]. The experiments will focus on performance, including data transfer and coding overhead, data storage and retrieval latency, and the effect of data caching. For the experiments, we will instantiate a few grid clusters in geographically different sites, such as different Brazilian cities and Spain.

We already implemented a prototype of the system and performed some simulations to evaluate OppStore. In one of the experiments, we simulated a grid of 100 clusters, each containing from 10 to 200 machines and machines mean idle times ranging from 30% to 70% of the time. We stored in this grid, 10k files, broken in 24 coded fragments, from which 12 are sufficient to reconstruct the file. When trying to retrieve 12 out of the 24 stored fragments, we achieved success rates of 94.8% with virtual ids and 86.3% whithout virtual ids. Consequently, using virtual ids effectively improved the file retrieval success rate.

We could not present other results and the detailed simulation setup here due to lack of space. We are currently preparing a paper with OppStore implementation details and experimental and simulation results for the CCGrid conference.

6. CONCLUSIONS AND FUTURE WORK

We designed OppStore to provide an infrastructure for reliable storage of files in opportunistic grids composed of shared machines, enhancing these systems by making available large amounts of free storage space. The middleware is designed to be self-organizing and fault-tolerant, allowing its operation in dynamic grid environments without the need for manual configuration. The use of virtual ids allows the system to automatically select clusters with larger amounts of storage space and higher machine availabilities for storing files. This should result in better usage of available storage space and improved file retrieval success rates. Also, by simultaneously using two id spaces, we removed the need for data movement when clusters changes their capacities.

We already have implemented prototype of OppStore, which we used to perform some simulations. We are now finishing a complete implementation of OppStore and its deployment over the InteGrade grid middleware. During the implementation, we had to deal with several design and implementation issues, such as data caching, security, and grid interfaces that we could not discuss in this paper due to lack of space. We are also evaluating other kinds of metrics that could be used to determine storage sites, such as network bandwidth and geographical proximity.

The next step will be to apply the concept of virtual ids for other

classes of applications. Although we are using OppStore as our initial evaluation platform, virtual ids could also be used outside the context of computational grids, for example in distributed peer-to-peer file systems and application-level multicast. We believe these applications will benefit from the cheap reconfigurations in the allocated id ranges for each node and the usage of multiple simultaneous virtual spaces.

7. REFERENCES

[1] G. Antoniu, L. Boug, and M. Jan. Juxmem: Weaving together the p2p and dsm paradigms to enable a grid data-sharing service. *Kluwer Journal of Supercomputing*, 2005. To appear. Preliminary electronic version available as INRIA Research Report RR-5082.

[2] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[3] M. Cai, A. Chervenak, and M. Frank. A peer-to-peer replica location service based on a distributed hash table. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 56, Washington, DC, USA, 2004. IEEE Computer Society.

[4] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and scalability of a replica location service. In *HPDC '04: Proceedings of the 13th IEEE Int. Symp. on High Performance Distributed Computing*, pages 182–191, Washington, DC, USA, 2004. IEEE Computer Society.

[5] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 2006. Accepted for publication.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM Press.

[7] R. Y. de Camargo, R. Cerqueira, and F. Kon. Strategies for checkpoint storage on opportunistic grids. *IEEE Distributed Systems Online*, September 2006.

[8] R. Y. de Camargo, A. Goldchleger, M. Carneiro, and F. Kon. The Grid architectural pattern: Leveraging distributed processing capabilities. In *Pattern Languages of Program Design 5*. Addison-Wesley, 2006.

[9] R. Y. de Camargo, A. Goldchleger, F. Kon, and A. Goldman. Checkpointing BSP parallel applications on the InteGrade Grid middleware. *Concurrency and Comp.: Practice and Experience*, 18(6):567–79, May 2006.

[10] M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, May 2002.

[11] I. Foster and C. Kesselman. *Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[12] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3), 2002.

[13] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured P2P systems. In *INFOCOM 2004: Proceedings of the 23th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 2253–2262, Hong Kong, March 2004. IEEE Computer Society.

[14] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. InteGrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16:449–459, March 2004.

[15] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43, New York, NY, USA, 2004. ACM Press.

[16] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 190–201, New York, NY, USA, 2000. ACM Press.

[17] M. Litzkow, M. Livny, and M. Mutka. Condor - A hunter of idle workstations. In *ICDCS '88: Proceedings of the 8th Int. Conference of Distributed Computing Systems*, pages 104–111, June 1988.

[18] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[19] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *IPTPS 2003: Proceedings of the Second International Workshop Peer-to-Peer Systems. LNCS (Vol. 2735)*, pages 68–79. Springer, October 2003.

[20] M. Ripeanu and I. Foster. A decentralized, adaptive replica location mechanism. In *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA, 2002. IEEE Computer Society.

[21] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM Symposium on Operating Systems Principles*, pages 188–201, New York, NY, USA, 2001.

[22] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, 2001.

[23] H. Shen and C.-Z. Xu. Hash-based proximity clustering for load balancing in heterogeneous DHT networks. In *IPDPS '06: Proceedings of the 20th IEEE Int. Parallel and Distributed Processing Symposium*, April 2006.

[24] I. Stoica, R. Morris, D. Karger, M. Kaashock, and H. Balakrishman. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[25] S. S. Vazhkudai, X. Ma, V. W. Freeh, J. W. Strickland, N. Tammineedi, and S. L. Scott. Freeloader: Scavenging desktop storage resources for scientific data. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 56. IEEE Computer Society, 2005.

[26] Z. Xu and L. Bhuyan. Effective load balancing in p2p systems. In *CCGRID '06: Proceedings of the Sixth IEEE Int. Symp. on Cluster Computing and the Grid*, 2006.