Design and Implementation of a Middleware for Data Storage in Opportunistic Grids

Raphael Y. de Camargo* and Fabio Kon
Dept. of Computer Science, Universidade de São Paulo, Brazil
Email: {rcamargo,kon}@ime.usp.br

Abstract

Shared machines in opportunistic grids typically have large quantities of unused disk space. These resources could be used to store application and checkpointing data when the machines are idle, allowing those machines to share not only computational cycles, but also disk space. In this paper, we present the design and implementation of OppStore, a middleware that provides reliable distributed data storage using the free disk space from shared grid machines. The system utilizes a two-level peer-to-peer organization to connect grid machines in a scalable and fault-tolerant way. Finally, we use the concept of virtual ids to deal with resource heterogeneity, enabling heterogeneity-aware load-balancing selection of storage sites.

1. Introduction

Opportunistic grids are a class of computational grids focusing on the usage of idle processor cycles from shared workstations [5, 10, 11] to execute computationally intensive parallel applications. But, besides processor cycles, applications from several areas, such as bioinformatics, data mining, and image processing, consume and/or produce large amounts of data, requiring an efficient data management infrastructure.

Current systems for data storage in computational grids usually store several replicas of files in dedicated servers managed by replica management systems [3, 4, 14]. These systems usually target high-performance computing platforms, with applications that require 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.

Meanwhile, shared machines from opportunistic grids often have large amounts of unused disk space. Combining the free disk space of a few hundred machines, we can easily achieve several terabytes of distributed storage space. Using these machines for storing data would improve resource utilization and enable a low-cost solution for data storage on institutions that have limited budget, for example in developing countries.

But data management on shared workstations requires a sophisticated middleware infrastructure. These machines are frequently turned off or restarted and can be used only when they are idle. A distributed storage system using these machines must ensure data availability in this highly dynamic and unstable environment. Also, computational grids may encompass tens of thousands machines, requiring the system to be self-organizing and highly scalable. Moreover, these machines can be highly heterogeneous.

In a previous work [7], we proposed *OppStore*, a middleware that explores the usage of a peer-to-peer overlay network for storage of read-only data in non-dedicated machines in the context of opportunistic grids. We also introduced the concept of *virtual ids* to deal with load-balancing and node heterogeneity. Using virtual ids, in addition to the original node id provided by Pastry [16], each node receives an extra virtual id, creating a virtual id space located on top of the Pastry id space.

In this paper, we describe the implementation of OppStore and its deployment in two current opportunistic grids. We also show obtained experimental and simulation results. We found that using OppStore with virtual ids improved stored data availability significantly and the delay for storing and retrieving data seemed acceptable for most cases.

2. Related Work

FreeLoader [18] 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

*Supported by a grant from CNPq, Brazil, process #141966/03-3.

sets of machines distributed across several clusters. Also, Freeloader does not consider load-balancing and machine availability when choosing storage sites.

JuxMem [1] implements a data sharing service for grid applications by combining 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 employ mechanisms for load-balancing and the availability properties 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, 14]. Some replica management systems [4, 14] 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 the global knowledge of replica locations, these systems have limited scalability. Also, the servers maintaining the replica locations are usually configured statically.

Cai *et al.* [3] built a replica location system using a distributed hash table 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 employs load balancing only for replica location queries and considers that all the servers have equal capacities. Our system uses a load-balancing technique that takes into account the heterogeneity of shared machines in grid clusters.

3. Peer-to-peer Networks and Virtual Ids

In OppStore, we extended the Pastry [16] protocol to include the concept of *virtual ids*. Pastry provides a peer-to-peer routing substrate that implements a distributed hash table. Pastry assigns to each node a random identifier, called Pastry id, from a range of valid identifiers, called Pastry id space. A message routed through Pastry is guaranteed to reach the node with the closest id to the message id. Pastry nodes maintain several tables, including the *leafset*, which contains a list of the node logical neighbors. At each step, the message is routed through nodes successively close to the target node, until it reaches a node containing the target node on its leafset. Then the message is redirected directly to the destination node.

An important problem with most DHTs is that issuing random ids to nodes results in some nodes being responsible for ranges in the id space $O(\log n)$ times larger than other nodes. Moreover, the heterogeneity of nodes is not considered. Finally, it should be possible to change the node ids dynamically to deal with changing environments.

To deal with these limitations, we assign to each node an additional identifier, called virtual id [7], using the same range of valid identifiers from Pastry, which we now call virtual id space. Virtual ids can be changed dynamically to reflect the heterogeneity of nodes and to adapt to dynamic environments where nodes constantly leave and join the network or change their characteristics. The *virtual partition protocol* redistributes the range of virtual space covered by a set of neighbors, assigning to each node responsibility for an id range proportional to its capacity. This capacity can be a function of one or more metrics, such as node availability, free disk space, bandwidth, and processing power.

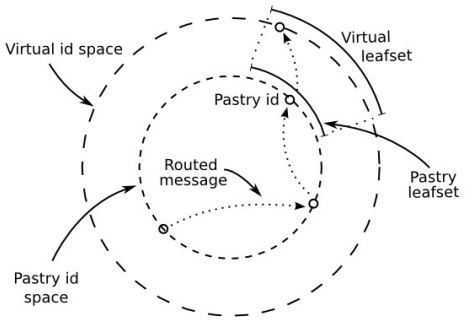

Figure 1. Virtual id space.

To allow routing in the virtual id space, each node maintains an additional table, called *virtual leafset*, shown in Figure 1. This table maps the original id space covered by the Pastry leafset of a node into the virtual id space, allowing the transition across these two spaces. Routing is performed using the Pastry algorithm, except for the last hop, where the node uses the virtual leafset to locate the destination node instead of the Pastry leafset. Consequently, routing in the virtual space requires the same number of hops as routing in the Pastry space.

Creating a virtual space adds only a small number of elements to Pastry, allowing us to perform heterogeneity-aware load-balancing with a small maintenance overhead.

4. OppStore Design

We designed OppStore as a middleware that enables the reliable and efficient storage of read-only data using the free storage space from idle machines in opportunistic grids. OppStore 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.

4.1. Middleware architecture

OppStore organizes grid machines as a cluster federation, where each cluster contains a Cluster Data Repository Manager (CDRM) and several Autonomous Data Repositories (ADR), one for each cluster machine. This organization should resemble the physical one, with machines from a single laboratory or institutional department located in the same cluster. CDRMs form a structured overlay network, using the Pastry [16] distributed hash table (DHT) algorithm. We use 160 bit ids to identify CDRMs and stored data uniquely. 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 2 shows the main OppStore components.

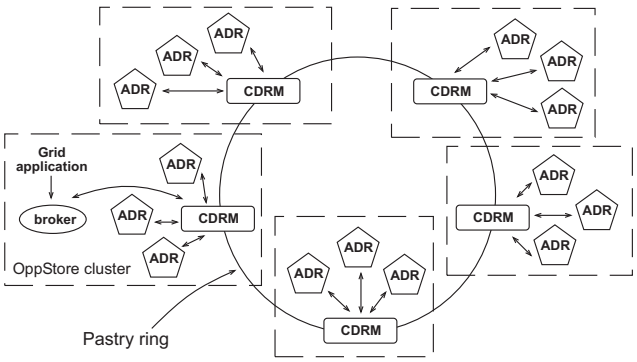

Figure 2. OppStore Architecture.

The 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.

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 highest bandwidth clusters, enabling efficient data retrieval.

4.2. Data storage and retrieval

Clients access the distributed storage system through the *access broker* library, 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 (IDA) [6, 13] 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 sets the *FFI id* as the secure hash from the fragments hashes and requests the cluster CDRM to store the FFI.

To download a file, the broker queries its cluster CDRM for the file FFI. The CDRM routes the request to the CDRM responsible for the file id, which returns the FFI. The broker then downloads the file fragments directly from the ADRs, checks the fragments integrity and reconstructs the file.

We should emphasize that file contents are not routed through the overlay network. They are transferred directly from the broker to the ADRs and vice versa. 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 across 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.

The *ephemeral mode* is used for data that requires high bandwidth and only needs to be available for a few hours. This class of storage would be used to store checkpointing [6, 9] 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 can use IDA or data replication to provide fault-tolerance.

4.3. Using virtual ids in OppStore

OppStore determines the storage location of fragments by routing the fragment identifier to the CDRM responsible for that identifier. To allow OppStore to consider the machine heterogeneities when selecting storage sites, we create a *virtual space* for the CDRMs in addition to the Pastry id space. This virtual space is much cheaper to maintain than the Pastry one and virtual ids can be changed with little overhead. To determine the CDRMs virtual ids, we define the capacity of each machine as its squared mean availability, which takes values from 0 to 1. Also, when the available storage space of a machine falls below a threshold, 1 GB in our case, the system linearly decreases the machine capacity according to the available space.

We define as the 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 larger amounts of available storage space. Actually, we can consider that each CDRM is responsible for managing the virtual spaces of all machines in its cluster.

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 less frequent. 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, transfers 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. Therefore, by using the Pastry and virtual spaces simultaneously, OppStore can adapt to dynamic conditions with very low overhead.

5. Implementation

The three main components of OppStore are: CDRMs, ADRs, and access brokers. In this section, we discuss their implementation and the management of stored data.

5.1. Cluster Data Repository Manager

CDRMs are responsible for managing their cluster ADRs, for selecting storage locations for data fragments, and for storing File Fragment Indexes (FFIs). Each CDRM maintains information about all ADRs from its cluster, including their id range, network address, state, capacity, and available storage space. When a CDRM receives a fragment storage request, it chooses an ADR. The probability of choosing an ADR is proportional to its capacity.

CDRMs are also responsible for the storage of FFIs. Since FFIs are small, containing only a few ADR addresses and fragment hash values, they are stored in the CDRM machine. To provide fault-tolerance, information contained in a CDRM is replicated in another r neighboring CDRMs. If a CDRM fails, an ADR from its cluster launches a new CDRM. Finally, CDRMs maintain a mapping of all fragments stored in the cluster ADRs, which is used to reconstruct the missing fragments in case of ADR departures.

5.2. Autonomous Data Repository

Each ADR is responsible for the storage of fragments corresponding to a range of ids proportional to its capacity. The ADR capacity is proportional to its mean availability. When a new node joins the cluster, it contacts the CDRM and provides its network address and available disk space. In the beginning, the CDRM uses the mean availability of cluster ADRs to evaluate the new ADR capacity. As availability values are collected from the ADR, the CDRM adjusts the ADR capacity accordingly.

The network address that ADRs provide to the CDRM when joining the cluster is used by access brokers to connect to those ADRs and transfer fragment contents. Different protocols can be used for connections from brokers to ADRs. In the case of TCP connections, the address can be the pair (IP address, port). For ADRs behind NATs, this address may contain the IP address of a proxy, which can be used to establish a TCP connection.

ADRs can be in three different states: *idle*, *occupied*, and *unavailable*. The ADRs are responsible for notifying the cluster CDRM about state changes. ADRs also send periodic keep alive messages to the cluster CDRM, allowing the detection of ADR failures. ADRs can be located in shared or dedicated machines. In case of shared machines, its owner can configure the machine to upload data only when it is idle or at any time, but possibly limiting the upload bandwidth when the machine is being used (occupied). Dedicated machines are always considered to be idle and, consequently, have a higher capacity.

5.3. Access Broker

The access broker is a library that allows applications to access the storage services of OppStore. It provides a C API containing methods for data storage and retrieval in synchronous and asynchronous modes. For data storage, calls to the broker can return (1) immediately, (2) after the broker finishes coding the data, or (3) after completing the file storage. In the first two cases, the application can provide

a callback function to the broker. Similarly, the library provides functions for data retrieval that return (1) immediately or (2) after finishing the complete data retrieval operation.

As described in Section 4.2, data stored in OppStore is coded into redundant fragments using the information dispersal algorithm (IDA) [6, 13]. IDA is an erasure code that allows one to code a vector U of size n, into $m+k$ encoded vectors of size n/m, with the property that one can regenerate U using only m of the $m+k$ encoded vectors. Analytical studies [15,19] show that, for a given redundancy level, data stored using erasure coding has a mean availability several times higher than using replication.

To reconstruct a file, the access broker only needs to download a subset of the stored fragments. Consequently, the broker can choose to download the fragments from the closest and/or fastest ADRs. The broker records the upload/download bitrates from previous transfers and uses this historical information to select the ADRs. We use a simple aging algorithm to estimate the bandwidth of a cluster, using some randomization to prevent that the same clusters are always selected. Also, as described later, OppStore has a caching mechanism to improve data retrieval performance.

Data integrity verification is enforced by calculating the SHA-1 secure hash of the file and fragment contents, storing then in the FFI. The system optionally provides data encryption for stored fragments. During storage and retrieval requests, the client can optionally provide a key, that the broker uses to encrypt and decrypt data. Note that OppStore does not provide key-management for data encryption, and the client is responsible for obtaining the key. We chose this approach because the key-management infrastructure is dependent on the deployment environment.

5.4. Data Management

OppStore caches the data necessary to reconstruct a file in the cluster in which the storage request was issued to improve data recovery performance. During the data storage process, when a CDRM returns to the access broker the addresses of remote ADRs, it also includes the address of one or more ADRs from its cluster. These local ADRs function as data caches if data is later requested from the same cluster. The access broker uploads a copy of some of the file fragments to these ADRs, and write their addresses in the FFI to allow tracking of the cached fragments. Caching for specific files can be disabled by the client if it knows that data is unlikely to be used in the local cluster.

To manage stored files, OppStore provides a lease to each file with a duration specified by the client. These leases can be renewed by clients at any time. During the leasing period, CDRMs check fragment availability periodically. When the leasing period ends, the file and fragments are marked as expired. Clients can try to renew a lease after it has expired, but the system offers no guarantees that it will be able to recover the file.

When analyzing data availability properties, we need to consider both temporary unavailabilities and membership changes. In the case of opportunistic grids, we consider a machine as unavailable if it is occupied or down. Several machine usage monitoring experiments [2, 8, 12] indicate that there are correlations in usage patterns in machine idle times and uptimes in a single laboratory. For example, during the night and weekends, machine idle time is normally higher than during the day. Also different environments, such as corporate and university labs, seem to have different usage patterns. Temporary unavailabilities can decrease the number of available machines to half in a period of a few hours. OppStore relieves the availability problem by placing most stored fragments in machines with higher availability and from geographically distant clusters.

Membership changes occur when ADRs or clusters leave the grid, causing the loss of stored fragments. These fragments should be replaced to ensure file survival. CDRMs are responsible for monitoring fragments stored in ADRs from its cluster. When an ADR leaves the system, its cluster CDRM sends a message to the CDRM storing the fragment FFI to mark that fragment as missing. If the number of available fragments is below a given threshold, that CDRM starts a fragment recovery procedure. We use a threshold because fragment replacement is an expensive operation that requires the reconstruction of the original file. Each CDRM also monitors its neighboring CDRMs to recover their cluster fragments in the case of cluster departures.

6. Case Studies

We analyzed the deployment of OppStore in two opportunistic grid systems, InteGrade [10] and Condor [17].

6.1. InteGrade

InteGrade [10] is a CORBA-based object-oriented middleware for opportunistic grids. It has a cluster federation structure, as shown in Figure 3. InteGrade stores grid data in OppStore, which returns an id for each stored file. Grid users access OppStore with a graphical interface and grid applications by using the access broker library.

InteGrade clusters have a machine designated as the cluster manager, usually a machine that is available most of the time. We instantiate CDRMs in the cluster manager, together with other InteGrade management modules. The *Global Resource Manager* (GRM) is responsible for scheduling and management of its cluster resources. The *Execution Manager* (EM) manages application execution and checkpointing. It maintains information about applications executing in the grid, including a mapping of check-

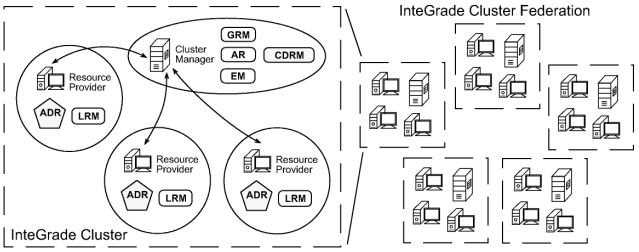

Figure 3. InteGrade's Architecture.

point files to OppStore ids. The *Application Repository* (AR) maintains meta-data about grid applications, including the mapping of their executable, input, and output files to OppStore ids. The *Global Security Manager* (GSM) is responsible for the security of an InteGrade cluster and manages user authentication and private keys.

ADRs are located in the same machines that provide resources to the grid, which we call resource providers. The main module in these machines is the *Local Resource Manager* (LRM), which is responsible for managing local resources and local processes from grid applications.

6.2. Condor

Condor [17] is an opportunistic grid middleware organized as a collection of clusters, called Condor pools. Machines sharing their resources run a *resource daemon*. Shared resource are registered in the *matchmaker*, which performs the matching between execution requests and resource providers. Grid users submit their application execution requests to a local *agent* module.

In a Condor grid composed of several condor pools, we could easily deploy OppStore placing the CDRMs in the same machines as the matchmakers and the ADRs in the machines running resource daemons. Consequently, the same machines that donate idle CPU cycles to the grid would also, optionally, donate unused storage space.

Finally, Condor allows applications to remotely access data stored on the machine from which the execution was submitted. Remote data access requires linking the application with a library that intercepts filesystem calls. This library could be modified to use OppStore, allowing transparent access to the distributed data storage functionality.

7. OppStore Evaluation

We evaluate OppStore using two methodologies: simulations and experiments in a controlled real grid environment. The objective of the simulations is to determine the availability of data stored in a a large-scale grid composed of non-dedicated machines. In the experiments, we measure the performance of OppStore, including the delay for data storage and retrieval in a wide-area grid.

We implemented the CDRM and virtual Ids protocols in Java, using FreePastry, an open-source implementation of the Pastry protocol, as the peer-to-peer substrate. For the ADR and access broker, we used C++ and Lua, allowing us to produce lightweight versions of these components.

7.1. Simulations

We performed simulations using a real OppStore implementation, replacing the ADRs and the access brokers by striped down versions written in Java. These ADR and access broker implementations simulate the process of data storage and retrieval, keeping track of "virtual" stored fragments. This enables the simulation of thousands of storage and retrieval operations in a large-scale computational grid.

We simulated a grid composed of 100 clusters, with the number of ADRs on each cluster randomly chosen as 10, 20, 50, 100, and 200. We used data from several machine utilization measurements [2, 8, 12] to define three different usage patterns, which are randomly assigned to each cluster. In the first pattern, the mean idletime is 60% during the day and 80% during the night and weekends. The second pattern has idletimes of 25% and 40%, and the third 40% and 70%, respectively. We used different values to measure machine usage during day and night periods to evaluate the effects of correlations in machine usage patterns. We simulated a one month period for two cases: clusters uniformly distributed across 24 timezones (*timezones-24*) and in a single timezone (*timezone-1*). This captures the two extreme cases of grids with machines distributed throughout the globe and geographically concentrated in a single timezone.

Data availability. We evaluated the rate of successful file retrieval using the machine usage patterns described above. We simulated the storage procedure for ten thousand files, including the routing of the fragments, and then performed the file retrieval steps, checking the number of fragments that could be recovered for each file. We stored fragments coded into 6 and 24 redundant fragments, comparing the usage of virtual ids with the Pastry algorithm using several redundancy levels. Finally, we considered that during the simulation there are no ADR and CDRM departures.

The graphs in Figure 4 show the ratio of successful file retrievals for files coded into 6 or 24 fragments and for clusters located in a single timezone or scattered throughout the globe. For each case, we considered several different redundancy levels by requiring a different number of fragments necessary to reconstruct the files. We simulated the case where fragments can be recovered only from idle machines.

As we can see, using virtual ids, we obtain significantly higher retrieval success rates. For instance, when requiring

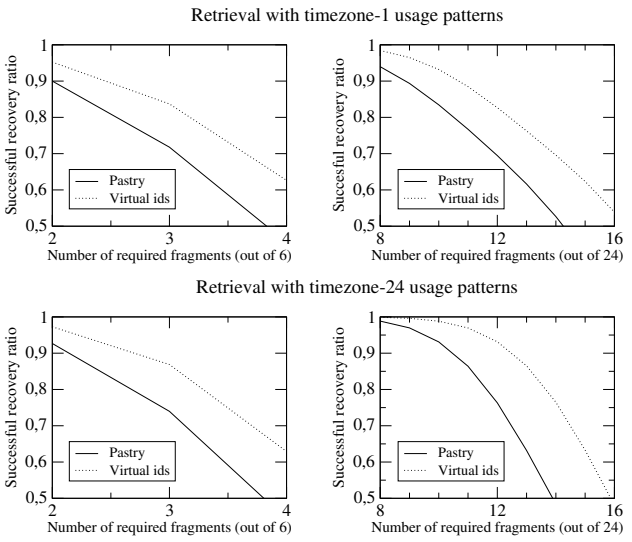

Figure 4. Successful file retrievals.

8 out of 24 fragments to reconstruct a file (replication factor of 3) and using the timezones-24 scenario, only 0.1% of the requests failed when using virtual ids compared with 1.2% when using the Pastry algorithm. When requiring 12 out of 24 fragments, the rate of failures in file retrievals increased to 6.8% and 23.7% respectively. Actually, we can see data retrieval improvements in all evaluated scenarios. It occurs because the system stores file fragments preferably in machines with higher availability.

We can also observe that scattering fragments throughout the globe improves retrieval rates when using higher redundancy levels. This occurs because we reduce the correlation of usage patterns among machines from different clusters.

To achieve good successful retrieval rates, we should use replications factors of at least 2, preferably 3, and break the file into more fragments. Also, when clusters are located in different timezones, the retrieval rate improves, since there is less correlation among machine usage times.

7.2. Experiments

We evaluated the usage of OppStore in an opportunistic grid environment. The experiments consist of storing and retrieving data in a small wide-area Grid composed of 5 clusters. The objective was to evaluate data storage and retrieval latency, including the data coding and decoding time.

We instantiated five grid clusters composed of commodity machines distributed in three Brazilian cities. São Paulo contained three clusters (sp1, sp2, and sp3) while Goiania (go) and São Luís (sl) contained a single cluster each. They are distant 900km and 3000km to São Paulo, respectively.

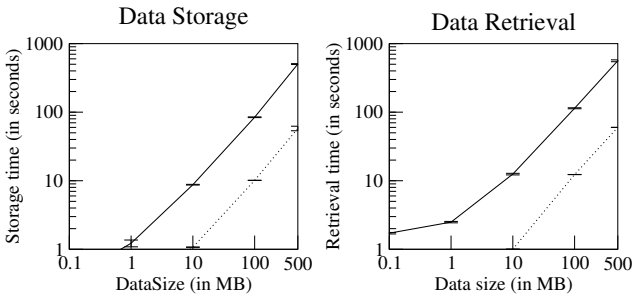

Figure 5. Data storage and retrieval.

The clusters are connected via the public Internet. We instantiated the broker in a 2GHz Athlon64 machine.

Data Storage. We stored files of several sizes. The broker split the file into 5 fragments, from which 2 were sufficient for data recovery. Since our experimental system had only 5 clusters, we forced the fragments to be stored on different clusters. Although forcing cluster selection may seem artificial, in a grid of hundreds of clusters the fragments would automatically be routed to different clusters.

The left-hand side graph in Figure 5 shows the delay for finishing the data coding process (dotted line) and to finish the storage process (solid line). When the broker finishes the data coding process, the application can continue its execution normally, while the broker performs the uploading of the fragments to the remote repositories. When storing a file of 100MB, the broker needs 12 seconds to code the file, while for 500MB the broker needs 60 seconds.

Completing the data transfer takes longer, requiring 560 seconds to transfer 1.25GB generated when coding a 500MB file. The total storage time is bound by the slowest connection, the São Paulo-Goiania one in our case, with a transfer rate of approximately 400kB/sec. But for most applications, the important delay is in data coding, which is much shorter and does not depend on network bandwidths.

Data Retrieval. We retrieved the files stored in the data storage experiment, with the broker downloading the two needed data fragments and reconstructing the original file. Retrieval times can vary widely depending on the repositories selected for fragment downloading. To track these variations, we performed retrieval experiments downloading the fragments from the two fastest repositories and from the two slowest repositories. These two scenarios should represent the extreme points regarding data retrieval times.

The right-hand graph in Figure 5 shows the time necessary to recover the files using the two fastest servers (dotted line) and the two slowest ones (solid line) relative to the file size. Retrieving data from the fastest servers (bandwidth of a few MB/s) required only 10 seconds to retrieve a

100MB file and 58 seconds for the 500MB one. Moreover, data decoding does not create a time overhead during data retrieval, since it occurs simultaneously with the fragment downloads. When using the slowest repositories, the data retrieval times are much higher, requiring about 500s to retrieve the file. But this should rarely occur, since the broker can always chose the fastest servers to download fragments.

8. Conclusions and Future Work

Using the disk space of shared machines scattered throughout the globe to perform reliable storage of data is a difficult task. OppStore addresses the complexities that arise from these highly dynamic and heterogeneous environments. We have shown that using only the idle periods of the shared machines to retrieve data, in realistic situations OppStore can provide data availabilities of 99.9% using a replication factor of 3 and 93.2% with a replication factor of 2. We achieved these good data retrieval properties by using the load-balancing properties of virtual ids.

Our middleware showed good performance when retrieving data from remote repositories. Since there are fragments located in several repositories, the broker can choose the fastest ones. When using data caching, access to data can be even faster. Data storage is slower, since the time required to store all the fragments from a file is bounded by the slowest connection. But the slower storage process can be improved by storing data incrementally as it is produced by an application executing in the grid or by calling asynchronous methods that perform storage in the background.

OppStore provides a viable low cost solution to the problem of data storage in opportunistic grids. It does not require the acquisition of extra hardware and can be easily deployed on existing opportunistic grid systems. We are now working into the further development OppStore. We intend to deploy OppStore for long periods in a real grid environment and analyze its usage patterns, providing us guidance to better improve it. For example, we may define efficient sharing policies to ensure fair use of storage resources.

References

[1] G. Antoniu, M. Bertier, E. Caron, F. Desprez, L. Boug, M. Jan, S. Monnet, and P. Sens. *Future Generation Grids*, chapter GDS: An Architecture Proposal for a Grid Data-Sharing Service, pages 133–152. Springer Verlag, 2006.

[2] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. *SIGMETRICS Performance Evaluation Review*, 28(1):34–43, 2000.

[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. IEEE Computer Society, 2004.

[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. IEEE Computer Society, 2004.

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

[7] R. Y. de Camargo and F. Kon. Distributed data storage for opportunistic grids. In *ACM/IFIP/USENIX Middleware Doctoral Symp.*, Melbourne, Australia, November 2006.

[8] P. Domingues, P. Marques, and L. Silva. Resource usage of windows computer laboratories. In *Int. Conf. on Parallel Processing Workshops*, pages 469–476, 2005.

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

[10] 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*, 16:449–459, March 2004.

[11] M. Litzkow, M. Livny, and M. Mutka. Condor - A hunter of idle workstations. In *Proc. of the 8th Int. Conf. of Distributed Computing Systems (ICDCS)*, pages 104–111, June 1988.

[12] M. W. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–284, 1991.

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

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

[15] R. Rodrigues and B. Liskov. High availability in DHTs: Erasure coding vs. replication. In *IPTPS '05: Revised Selected Papers from the Fourth International Workshop on Peer-to-Peer Systems*, pages 226–239. Springer-Verlag, 2005.

[16] 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.

[17] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[18] 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.

[19] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 328–338. Springer-Verlag, 2002.