Efficient Maintenance of Distributed Data in Highly Dynamic Opportunistic Grids*

Raphael Y. de Camargo
School of Arts, Sciences and Humanities
Univ. of São Paulo, Brazil
rycamargo@usp.br

Fernando Castor Filho
Department of Computing and Systems
Univ. of Pernambuco, Brazil
fjclf@dsc.upe.br

Fabio Kon
Department of Computer Science
Univ. of São Paulo, Brazil
kon@ime.usp.br

ABSTRACT

Opportunistic grids are a class of computational grids that can leverage the idle processing and storage capacity of shared workstations in laboratories, companies, and universities to perform useful computation. OppStore is a middleware that allows using the free disk space of machines from an opportunistic grid for the distributed storage of application data.

But when machines depart from the grid, it is necessary to reconstruct the fragments that were stored in that machines. Depending on the amount of stored data and the rate of machine departures, the generated traffic may make the distributed storage of data infeasible. In this work we present and evaluate a fragment recovery mechanism that makes viable to achieve redundancy and large data scale in a dynamic environment.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

General Terms

Design, Performance, Reliability

Keywords

Grid computing, distributed storage, data redundancy

1. INTRODUCTION

Computational grids [6, 5] coordinate geographically dispersed resources to allow the execution of computer-intensive applications. Opportunistic grids, such as InteGrade [7], are a class of computational grids that can leverage the idle processing and storage capacity of shared workstations in laboratories, companies, and universities to perform useful computation. Several classes of parallel applications manipulate large amounts of data and need a data storage and management infrastructure. OppStore [4] is a middleware that allows the storage of application data using the unused disk space of non-dedicated machines in the context of opportunistic grids.

There are several challenges inherent to ensuring the availability of distributed data. First, it is necessary to use the machine resources only when they are idle. Also, those machines are frequently turned off during the night and weekends, reducing the time available for data transfer. Another problem is that machines may leave the Grid unexpectedly, and data stored in these machines is permanently lost. Moreover, if a machine owner shared some of its resources with the Grid and later needs the disk space back, the OppStore files will be removed immediately. To solve these problems, OppStore encodes files into redundant fragments and distributes those fragments across different grid clusters. This allows files to be retrieved even if some of the fragments are not available. Additionally, it implements a mechanism to reconstruct lost fragments from existing ones, to maintain the desired level of redundancy and to prevent files from becoming unavailable. Furthermore, to improve scalability, OppStore organizes the grid machines into a federation of clusters, where each cluster is typically composed of machines from a single laboratory or department of an institution. The clusters are connected by a peer-to-peer network based on the Pastry [14] distributed hash table (DHT).

As noted by other researchers [2], designing and implementing a system capable of providing high redundancy for applications that store large amounts of data in dynamic environments is a difficult task. In this scenario, the bandwidth required to maintain the data in the case of node departures becomes impractical. Nevertheless, to build highly available and reliable opportunistic grids that manipulate large amounts of data, these difficulties have to be overcome.

In this work, we present a data maintenance mechanism for OppStore that operates in highly dynamic environments with low-overhead. The efficiency of the mechanism can be achieved due to the following characteristics of OppStore: (1) organization of machines into a federation of clusters and (2) decoupling data storage location from its identifier. The organization of machines in a federation of clusters allows dealing with machine joining and departing locally in the cluster. The decoupling of data storage location from its identifier means that when a machine joins or leaves the grid, data stored in its neighbors do not need to be migrated to correct the placement of data.

*This research is supported by CNPq/Brazil, grants #481147/2007-1 and #550895/2007-8.

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

Machine departures are the main cause for data management traffic, since fragments stored in the machines have to be reconstructed to maintain the redundancy level. The fragment reconstruction operation is expensive, since it is necessary to reconstruct the original file for every lost fragment. Consequently, we need to prevent its execution whenever possible. The strategy we propose in this work is to keep an extra copy of each stored fragment in another machine from the same cluster where it is stored. Thus we can regenerate the fragments lost due to machine departures from the other copy of the fragment. Our strategy is based on the assumption that free disk storage space and intra-cluster network bandwidth are less scarce than the inter-cluster network bandwidth. We consider this a reasonable hypothesis for opportunistic grids.

We evaluated the proposed mechanism measuring the obtained file availability and the amount of inter-cluster and intra-cluster bandwidth used for data maintenance, comparing this mechanism with other alternatives to reduce bandwidth. The main contribution of this paper is to present and evaluate a mechanism that makes viable to achieve redundancy and large data scale in a dynamic environment.

2. RELATED WORK

There are a number of works in the area of distributed data storage in peer-to-peer networks and computational grids. PAST [13], CFS [3], and pStore [1] are peer-to-peer distributed storage systems built over a DHT infrastructure, such as Pastry [14]. To improve availability, PAST stores multiple replicas of the files and CFS and pStore break the files into fragments and store several replicas of those fragments. The main differences to our system are that PAST, CFS and pStore organize all machines at a single level and store data directly in the machines that perform message routing. Consequently, each node arrival and departure requires that large amounts of data are transfered to compensate for changes in machine organization, suffering from the problems described by Blake *et al.* [2].

OceanStore [8] creates a global-scale persistent storage. It codes the data into redundant fragments and distributes the fragments amongst several machines. Differently from our work, it focus on dedicated machines, which means that data reconstruction seldom needs to be performed.

PeerStore [9] is also a peer-to-peer distributed storage system. Like our work, it decouples fragment location from its identifier, storing the file index and fragments separately. This eliminates the maintenance due to data misplacement caused by node joinings and departures. Notwithstanding, it recovers lost fragments caused by departures in a lazy way, compromising the immediate recovery of stored files. This characteristic makes it less suitable for very dynamic environments, such as opportunistic grids.

A common technique for computational grids is using data replication in conjunction with a replica location system [12]. These systems store the grid application data on dedicated storage servers and are not targeted at opportunistic grids.

3. OPPSTORE

Computational grids are commonly organized as a federation of clusters [6, 7], where a computational cluster is composed of physically close machines, for example, in a single laboratory or department. OppStore [4] organizes the machines in the same fashion, allowing its deployment over existing grid middlewares. We consider that each cluster has a management machine, where we instantiate the CDRM (*Cluster Data Repository Manager*), the module responsible for manages the machines within the same cluster. The other machines provide free space for storage of application data and execute the ADR (*Autonomous Data Repository*) module. CDRMs are organized in a *peer-to-peer* network, structured as a DHT, and leveraging Pastry [14] as substrate. The DHT is used to locate the ADRs where OppStore will store the files and to retrieve those files later. Figure 1 shows the OppStore architecture.

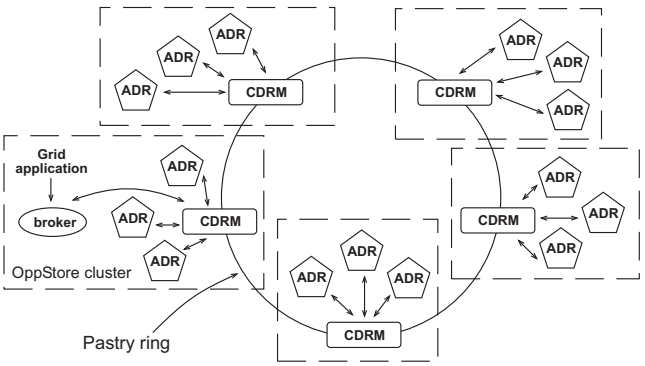

Figure 1: OppStore architecture.

A two-level architecture facilitates grid management, since machine departures and joins can be dealt with locally within the cluster. Information about nodes leaving and joining the cluster only need to be propagated to other clusters when they pertain to the machine executing the CDRM. We assume that this will be an infrequent event.

3.1 Data Management

OppStore allows grid applications to store files using two storage methods: (1) ephemeral and (2) perennial. In the ephemeral mode, two replicas of the file are stored in the machines from the cluster where the storage request was issued. This mode is used for data that will be used in the same cluster where it is stored and within a few hours at most. A typical use is for storage of application *checkpoints*.

The perennial mode is for data that will be stored for periods of time longer than a few hours. In this mode, OppStore encodes the files into redundant fragments, using an optimized version [10] of the Information Dispersal Algorithm (IDA) [11]. This coding generates n fragments from which any $k \leq n$ are sufficient to reconstruct the original file. Total fragment sizes is n/k times the original file size.

OppStore transfers the fragments to ADRs at different clusters and constructs a File Fragment Index (FFI), containing the storage address of each fragment and some extra information. The contents of the FFI are then hashed to generate an unique identifier for the FFI. Finally, a message containing the FFI is routed in the DHT for storage in the CDRM responsible for the FFI identifier. Updating a stored file would be possible, but we consider that stored information is mostly application input and output data.

Data coding is fast and, consequently, the time needed to store files is normally limited by the network speed. When using the public Internet to connect the clusters, files of a

few hundred MBytes can be stored in a reasonable time [4].

Although the use of erasure coding improves data availability significantly [15], when a machine leaves the grid, fragments are lost and need to be reconstructed. Moreover, a machine owner may need the disk space that it shared with the grid, requiring OppStore to immediately remove the fragments from the machine. Cluster departures are also possible. This case would be equivalent to all machines from the cluster leaving the grid simultaneously.

It is difficult to know if a node departure is temporary or definitive. Moreover, we use heartbeats as a way to detect node departures and, consequently, false departure detections can happen in the case of transient network failures. To minimize the impact of transient departures and false node departure detection, we delay the beginning of the reconstruction until a significant number of fragments are lost, in the hopes that some of the nodes have left the grid only temporarily or that the departure was a false detection. We call *reconstruction threshold* the minimum number of fragments that need to be available.

Another important traffic generator is that when machines from a DHT join or leave the system, it is necessary to correct data misplacement due to the reorganization of the identifier space [2, 9]. Since OppStore decouples the fragment storage location from its identifier, transfers are not necessary during machine joins and departures [4].

4. RECONSTRUCTION PROTOCOL

We designed a reconstruction protocol for lost fragments that works in all cases and defined several optimizations for this standard protocols to reduce the reconstruction overhead. The reconstruction protocol is given by:

1. The CDRM from the cluster where a machine departed notify the CDRMs containing the FFIs of the fragments stored on that machine.

2. For each notification a CDRM receives, it verifies if the reconstruction threshold was reached. If affirmative, the CDRM downloads the fragments necessary to reconstruct the original file directly from the ADRs.

3. The CDRM reconstruct the lost fragments from the file and routes a message for each generated fragment requesting an ADR address to store the fragment.

4. The CDRM transfers the generated fragments to their storage sites and update the FFI of the files with recovered fragments with the new storage sites.

To estimate the cost of the reconstruction process for the lost fragments of each file, we consider the number of messages that the system routes in the peer-to-peer networks and the number of bytes transfered directly from the broker to the ADRs and vice-versa. If we define n as the number of fragments generated per file, k the number of fragments necessary to reconstruct a file, r the reconstruction threshold and N the number of cluster in the grid, the cost of each reconstruction step is given by: (1) 1 message, (2) $fileSize$ bytes, (3) $n - r$ messages, and (4) $(n - r)/k \times fileSize$ bytes. Consequently, the protocol requires transferring $(1 + (n - r)/k) \times fileSize$ bytes of data and routing $1 + n - r$ messages in the peer-to-peer network.

There are some approaches to reduce the communication cost of the reconstruction protocol. For example, it is possible to decrease the reconstruction threshold, which reduces the number of times the reconstruction protocol is started and regenerates more fragments in a single execution of the protocol. The disadvantage is that file availability is reduced, since less fragments are available while the reconstruction threshold is not reached. Another alternative is to increase the redundancy level, providing higher file availabilities and less calls to the reconstruction protocol, but file storage becomes more expensive.

Finally, OppStore can *cache a copy of the stored file* in the node from the data storage request was performed at no cost by keeping a link to the file in the local file system. Consequently, it is possible for the CDRM containing the FFI to ask directly for this node (if it is idle) to reconstruct and send the fragments again. We evaluate all these optimizations in the simulations.

4.1 Local Fragment Copy

The protocol for reconstruction of fragments just presented can reconstruct lost fragments in all scenarios, but requires large amounts of communication resources.

OppStore organizes machines in a federation of clusters and decouples fragment storage location from its identifier by using the File Fragment Indexes (FFIs). Consequently, we can prevent starting the fragment reconstruction process in the majority of cases by keeping, for each fragment stored in cluster, an extra copy of that fragment in another machine from the same cluster. When a machine departs, the CDRM can immediately regenerate each of the lost fragments using the copies stored in the cluster. In other words, machine departures are treated locally in the cluster.

In this approach, we have to double the required storage space and put extra strain in the local networks, but now fragments are lost only when the two machines holding the copies of a fragment leaves the grid almost simultaneously. If we combine the local fragment copy with a reconstruction threshold, we eliminate the need for the reconstruction protocol in the vast majority of the cases. Moreover, keeping local fragment copies puts no extra burden in the inter-cluster connections, which normally have a more limited amount of bandwidth than local networks. Consequently, if storage space and local network bandwidth are not scarce, we can use this approach to solve the data maintenance problem.

5. EVALUATION

We performed simulations using our implementation of OppStore to evaluate the following points: (1) if it is feasible to deploy OppStore in a typical dynamic opportunistic grid environment, (2) how much bandwidth is needed to keep the file availability level in the presence of machine departures, and (3) which is the best approach to reconstruct the fragment for different scenarios. Since we needed to evaluate the behavior of OppStore in large-scale grids in several scenarios, we consider that using simulations was a better choice.

5.1 Simulation Parameters

We simulated an opportunist grid composed of 30 clusters of desktop machines from universities and institutions. The number of machines on each cluster was randomly chosen as 10, 20, 50, 100, and 200.

To evaluate file availability, we defined three different usage patterns, which are randomly assigned to each cluster. In the first pattern, the mean idle time is 60% during the day and 80% during the night and weekends. The second pattern has idle times of 25% and 40%, and the third 40% and 70%, respectively. We consider that clusters are uniformly distributed across 24 timezones.

Fragment losses can occur with machine departures and when the owner of a machine demands back the shared disk space. In both cases we consider that all the fragments in the ADR are lost and we evaluated a range of loss rates to determine the expected overhead of the reconstruction protocol. To keep the number of machines stable, we consider that when a machine leaves the grid, another one joins the grid immediately in a random cluster. Finally, we consider that the network is reliable and that messages are always delivered.

5.2 Data availability

In this experiment, we evaluate the availability of files stored in OppStore in the presence of machine departures. We stored 3000 files and simulated a one month period with a machine departure rate of 10% per day, running the fragment reconstruction protocols. At the end of this period, we requested the recovery of all stored files and measured the percentage of the files that could be recovered.

Table 1: Mean data availability.

IDA	t=9	t=12	t=15	fc
k=6, n=12	0.45133	-	-	0.84733
k=6, n=18	0.73833	0.87500	0.99733	1.00000
k=6, n=24	0.88100	0.92933	0.95867	1.00000

Table 1 shows the mean availability for several IDA configurations and threshold values and for the fragment copy strategy. As we increase the threshold, the mean availability of the stored files also increases, since the fragment recovery protocol is executed more often. But, the availability when using fragment copy is always higher, since fragments are immediately recovered from their copies.

5.3 Network usage

In this experiment, we measure the network traffic generated for each strategy for different ADR fragment loss rates. We stored 3000 files of different sizes in OppStore, with a total size of about 65GB. We simulated a period of 1 month, where ADRs depart or lose their stored fragments at different rates per day.

Figure 2 shows the generated bandwidth used to store the files and recover the lost fragments for different ADR fragment loss rates, using IDA(6,12) with threshold 9, IDA(6,24) with threshold 9, IDA(6,24) with threshold 15, and IDA(6,12) with the fragments copy strategy.

When not using fragment copy, the required bandwidth increases linearly with the departure rate. But even when considering a departure rate of 1.25% of the nodes per day, the used bandwidth is about 130GB to store the files and 150GB to maintain those files. Consequently, in a single month, an amount of data comparable in size to all of the stored data is transfered between clusters. Also, for higher departure rates, we see that increasing the replication level

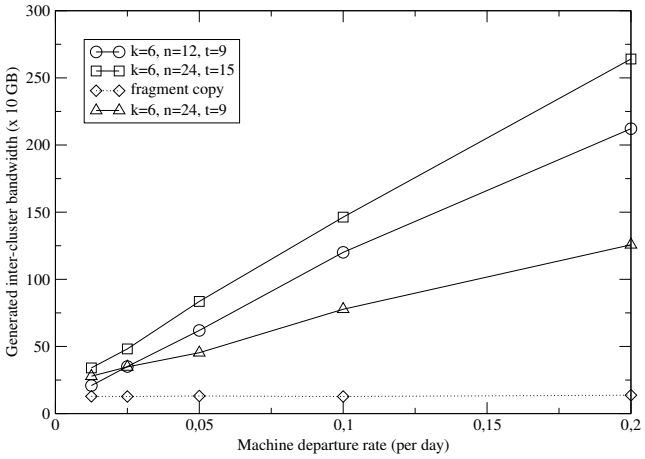

Figure 2: Bandwidth for fragment reconstruction.

lowers the bandwidth necessary to maintain the data, for example, when we increase the number of fragments to 24. Reducing the reconstruction threshold also lowers the used bandwidth. But as we saw in Section 5.2, file availability decreases when we decrease the reconstruction threshold.

When using the fragment copy, for all ADR departure rates, the inter-cluster bandwidth is constant and lower than the other strategies. This occurs because this bandwidth is used only for the transfer of fragments during file storage. Consequently, when considering the inter-cluster bandwidth used, the fragment copy strategy is very efficient.

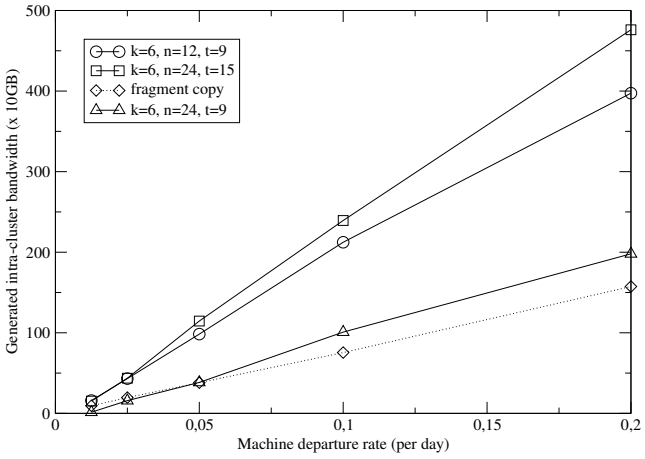

Figure 3: Local Bandwidth during reconstruction.

Figure 3 shows the generated bandwidth in the local networks inside the clusters using the same scenario above. It is interesting because it shows that the usage of local network bandwidth for the fragment copy strategy is also lower. This result can be explained if we remember that, in the reconstruction protocol without fragment copy, every time the fragments are transfered in the inter-cluster network, they are also transfer in the local networks of two different clusters. Consequently, besides the load in the inter-cluster network, a high load is also put in the local networks.

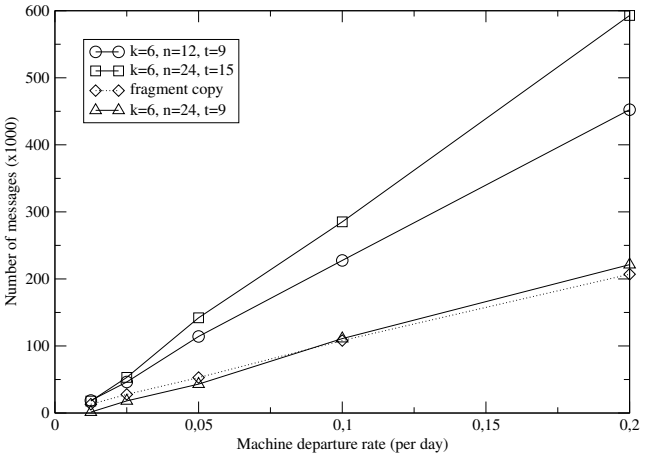

Figure 4: Number of messages exchanged.

Consequently, even in the usage of bandwidth of local networks, the fragment copy strategy performs better. The exception is for the lowest departure rates, the need to keep two copies of each fragment increases the local network usage. Moreover, when we analyze the highest departure rate of 20%, the total generated local bandwidth is about 1500GB in a month, giving 50 GB per day in all 30 clusters, which is about 1670 MB of traffic per day in the local networks of each cluster. Considering that those networks have a bandwidth of at least 100Mbps, this overhead is very reasonable.

Finally, Figure 4 shows the number of messages that are routed in the peer-to-peer network for the different reconstruction strategies. When using fragment copy, every departure of a machine causes the generation of 1 message per lost fragment to notify the change in the storage address of the fragment. The number of messages generated when using the local fragment copy is close to the use of IDA(6,24) and a threshold of 9.

6. CONCLUSIONS

In this paper, we showed that it is possible to store large amounts of data with high redundancy in a highly dynamic environment. For this, we presented a data maintenance protocol that takes advantage of the organization of machines into a federation of clusters and the decoupling of data storage location from its identifier. We showed that, in realistic scenarios, this protocol uses reasonable amounts of intra-cluster bandwidth and very low amounts of inter-cluster bandwidth.

As future work, we will deploy OppStore for long periods in an opportunistic grid and monitor the fragment recovery mechanism, validating the results obtained in the simulations. Another important point is to consider limiting the network usage when there are machine owners utilizing the network for their tasks. If users notice a significant loss in performance, they will be unwilling to share their resources.

7. REFERENCES

[1] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pStore: A secure peer-to-peer backup system. Technical Report MIT-LCS-TM-632, MIT LCS, October 2002.

[2] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: pick two. In *HotOS'03: Proc. of the 9th Workshop on Hot Topics in Operating Systems*. USENIX, 2003.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 202–215, New York, NY, USA, 2001. ACM Press.

[4] R. Y. de Camargo and F. Kon. Design and implementation of a middleware for data storage in opportunistic grids. In *CCGrid '07: Proc. of the 7th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*. IEEE Computer Society, 2007.

[5] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *NPC 2005: IFIP International Conf. on Network and Parallel Computing*, pages 2–13, Beijing, China, 2005.

[6] 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):237–246, 2002.

[7] 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.

[8] 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 '00: Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, 2000.

[9] M. Landers, H. Zhang, and K.-L. Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. In *P2P '04: Proc. of the 4th Int. Conf. on Peer-to-Peer Computing*, pages 72–79. IEEE Computer Society, 2004.

[10] Q. M. Malluhi and W. E. Johnston. Coding for high availability of a distributed-parallel storage system. *IEEE Transactions Parallel Distributed Systems*, 9(12):1237–1252, 1998.

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

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

[13] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proc. of the 8th ACM Symposium on Operating Systems Principles*, pages 188–201. ACM Press, 2001.

[14] 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 Int. Conference on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, 2001.

[15] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Proc. of the 1st International Workshop on Peer-to-Peer Systems*, pages 328–338, 2002.