On-demand Grid Storage using Scavenging

Sudharshan Vazhkudai
Computer Science and Mathematics Division
Oak Ridge National Laboratory
vazhkudaiss@ornl.gov

Abstract

Increasingly scientific discoveries are driven by analyses of massively distributed bulk data. This has led to the proliferation of high-end mass storage systems, storage area clusters and data centers as storage fabric elements for the Grid, offering excellent price/performance ratio and good storage speeds, but increasing maintenance and administrative costs. A promising alternative then, is to harness the collective storage potential of individual workstations much as we harness the idle CPU cycles due to the affordable economics in aggregating commodity storage and low usage to available space ratio. However, such aggregated commodity storage is prone to volatility, machine failures, performance concerns and trust issues. In this paper, we address several of the aforementioned issues and present our design on the construction of scalable aggregated commodity storage through scavenging; providing availability through aggressive replication; revering user autonomy and his authority to reclaim space; and enabling Grid access to such storage.

Keywords: *Grid Storage, Serverless Filesystems, Scavenging.*

1. Introduction

Grids are often quoted to have grown out of the traditional supercomputing arena—the Computer Center model—wherein users, limited by the computing power within a cluster/domain, needed mechanisms to perform computational operations on clusters in other domains [FK98, LEGION03]. Numerous solutions have been designed to present a collective (domain) computing potential to the Grid ranging from those built for tightly-coupled Beowulf style clusters [Merkey94] to loosely-coupled, idle workstations [LLM88]. These solutions serve as excellent fabric elements for the Computational Grid. Thus, a fundamental change is occurring in the computing landscape. Proprietary systems are being replaced with commodity clusters, delivering new levels of performance and availability at dramatically affordable price point.

Increasingly though, scientific discoveries are driven by analyses of massively distributed bulk data. This has led to the proliferation of high-end mass storage systems [CW95, DPSS03, TWD95], storage area clusters and data centers (IBM, Panasas, HP) as storage fabric elements for the Grid. These systems offer excellent price/performance ratio, good storage speed and access control, support for intelligent parallel file systems, optimization for wide-area bulk transfers and reliable storage. Several of them have been successfully demonstrated and are in use in major multi-institutional Grid efforts including the TeraGrid [TeraGrid04], Earth Systems Grid [ESG03], and DOE Science Grid [DSG03].

However, high-end storage also comes with increasing deployment/maintenance/administration costs, specialized software and central points of failure. Further, the cost and specialized features prohibit their wider acceptability and limit them to a select few research laboratories and organizations. If grids are to become prevalent and grow beyond the confines of a few organizations, exploiting commodity fabric features is absolutely essential.

A promising alternative then, is to harness the collective storage potential of individual workstations much as we harness the idle CPU cycles (Condor [LLM88], SETI@Home [SETI03]). The potential of collective commodity computing has been demonstrated time and again, outpacing supercomputers in their ability to deliver sustained high-throughput computing required by several current applications. Such an approach for storage is desirable and made feasible due to the following reasons.

First and foremost, the economics of buying gigabytes of more storage is increasingly becoming affordable so that even ordinary user desktop workstations are equipped with tens of gigabytes. Second, recent studies in corporate LAN settings indicate that up to 50% of disk space is unused [ABC+02]. This suggests that space usage to available storage ratio is significantly low thereby justifying the aggregation and use of individual workstation storage (a storage-equivalent argument for "*most computers are idle for substantial amounts of time justifying idle cycle stealing*"). Finally, these workstations—in increasing numbers—are online most of the time and, therefore, even a meager contribution—where *Contribution << Available*—from each workstation could result in collective staggering aggregate storage. For instance, in Computer Science and Math Division (CSMD, primary author's home division in Oak Ridge National, ORNL) alone up to 600 computers is estimated to be online at any given time. A small contribution of 2 GB per workstation could easily amass over a terabyte of storage.

Yet, there is reluctance (justified) in espousing such an approach for Grids due to the following reasons. First, desktop workstations—due to their sheer nature of individual ownership and lack of central-storage like control—are prone to the vagaries of volatility. Machine failure rates and availability can vary unpredictably and so does access to data contained in them. Second and equally important, is the question of trusting datasets on user desktops which gives rise to the possibility of data corruption and malicious users—a problem not as severe as in peer-to-peer storage since individual workstations in question (in our case), after all, belong to the same administrative domain and are ultimately accountable. Third and of significance to Grid applications, is the performance that can be derived out of such aggregate storage. Despite the high maintenance and administrative costs, a factor that attracts the Grid community to high-end storage and data centers is their ability to deliver sustained high-throughput for data operations.

In this paper, we present our design and work in progress on:

- Aggregating storage in organizational domains constructed by "*scavenging idle storage space*" from individual workstations, thereby forming Grid Storage Service Providers (SSP).
- Enabling on-demand Grid access and address issues involved in data-intensive application requirements posed to such cumulative commodity storage.
- Minimizing, as much as possible, the differences perceived by a Grid client in terms of accessing aggregated commodity storage.

Thus, in order to achieve the aforementioned goals we need to investigate several of the following issues whose design we present in detail in the subsequent sections. First, our solution needs to be *scalable* to thousands of transient desktop workstations. Second, we need to address *reliability and availability* of data through replication. This is particularly essential since machines can join and leave the pool at random. Third, our approach has to ensure *data correctness and security*. Fourth, the scavenging mechanism needs to be *transparent and non-invasive* to the workstation user, respecting his control, local autonomy and authority to reclaim the space at will. Fifth, we need strategies to address performance issues in data aggregation, replication, and transport and yet not compromising flexibility. Finally, we need to support several *Grid specific requirements* namely data/storage management activities (distributed space reservations, distributed pinning abilities, etc.), data transfer protocol agnostic features and the like.

2. Use Cases

One way to use commodity storage is in the Data Grid replication process. Data Grids achieve high-availability by replicating bulk data—several gigabytes and even terabytes—across storage clusters in the participating domains. For example, ORNL's Spallation Neutron Source (SNS) will generate several terabytes of data once operational in 2006, which will need to be accessed by geographically dispersed users. Or, several high-energy physics experiments [GriPhyN02, HJS+00, Holtman00, HSS00, LIGO02] have agreed on a tiered Data Grid architecture in which subsets of data are replicated across the tiers [DataGrid02]. Therefore, any particular dataset is likely to have replicas located at multiple sites. As we mentioned earlier, using aggregated commodity storage alternatives, regular organizations—not just specialized, select sites—can become part of this infrastructure. Aggregated, loosely connected storage can be used to store hundreds and thousands of replicas in a transparent fashion, becoming part of the replica cataloguing infrastructure of the Data Grid.

Alternatively, commodity storage could also be used to stage data before they are moved to a costly, high-end storage which may not have been available. For instance, in Grid systems, data access and jobs are coordinated with the use of advance reservations to storage resources, schedulers and the like. These high-end resources are almost always busy with long queues of pending requests. The aggregated storage cloud could be used in such cases to stage datasets—bringing data closer to the computation from remote locations—which will eventually be moved onto high performance storage for processing.

Data referred to above is mostly for computations. There exists another class which we refer to as "*in transit*" data. An instance of transit data is results from computations on-the-way to their final destinations, which are often transferred hop-by-hop—using movement tools such as Kangaroo [TBS+01] (from Condor)—through several intermediate storage locations due to current unavailability of the end resource. These datasets—though on the constant move—can also be substantial in size. In this setting, aggregate commodity storage can be used to store these ephemeral datasets in Grid environments, offering an inexpensive alternative to costly high-end storage. Even in the presence of high-end storage clusters, transit data can be exclusively stored only in commodity workstations leaving aside expensive storage to data meant for computation which pose stringent I/O demands.

3. Design Choices and Assumptions

Scalability: The storage resource management environment is intended to support several hundreds or even thousands of workstations within an administrative domain, handling data access requests from hundreds of clients as well.

Commodity Components: Our storage scavenging system is built from inexpensive, commodity workstations where failure is the norm and not an exception. The quality (commodity

workstations) and quantity (scalability numbers above) indicates that our system should have strong support for fault monitoring, notification and recovery.

User Autonomy: Our system is based on space contribution from individual users and revolves around the premise that the user ultimately has the right to reclaim the space. Thus, our design needs to reflect this guiding principle with support for dynamic space shrinkage and growth. Both building on commodity components and respecting user autonomy means we need mechanisms to support availability.

Connectivity & Security: We assume a well connected corporate LAN setting but not high-speed communication environments expected by parallel file systems. Further, we assume a fairly secure environment and no malicious intent in the workstation user's part. This is a luxury which a lot of p2p storage systems cannot afford due to cross-administrative collaborations. Yet, we need some basic security in place.

Heterogeneity: User desktop workstations come in all flavors ranging from operating system diversity to machine characteristics—CPU speeds, disk speeds, network bandwidths—to varying temporal loads. Our architecture will need to accommodate such a diverse mix and exploit the functional differences therein.

Data Properties: The datasets in question are huge, immutable files (write once read many) ranging from several hundred megabytes to several gigabytes. These are large by traditional standards but are the norm for Grid and data-intensive applications. We intend the system to be used to store several hundreds of such large files.

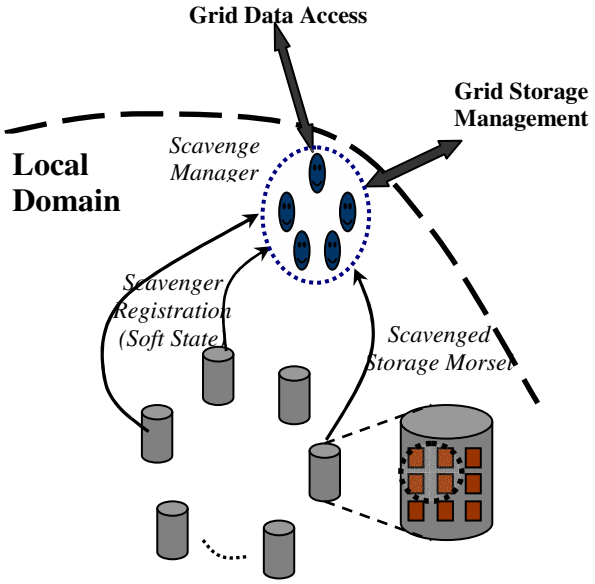

Figure 1: An architecture for aggregating idle workstation storage space using scavenging.

Grid Awareness: We assume that our local storage management solution would be presented for use for the Grid, receiving numerous requests for data access from multiple remote clients. Typical Grid applications impose stringent response time requirements on file accesses and demand high, sustained data transfer throughput. Further, Grid applications require support for data staging, space reservations, etc., which need to be translated to local storage management operations.

4. Architecture

Our solution constructs aggregate storage from numerous commodity workstations to be presented as Data Grid fabric elements. In designing the architecture we are faced with two key considerations namely *Decentralization* and *Scalability*. In this section we discuss our solution in the context of these two issues.

4.1. Soft-State Registration

Our design (Figure 1) comprises of individual workstation users volunteering to contribute a piece of their storage space—called *Morsel*—for certain duration of time using a *Scavenger process* to a *Scavenge Manager Group*. The scavenger and manager coordinate with each other using a scalable soft-state registration protocol. Each scavenger announces its arrival into the environment by registering itself with a manager and thereafter constantly updating the manager with keep-alive messages. Such a protocol has the following advantages in our context. First, our system is entirely composed of numerous commodity workstations owned by individual users and is thus prone to failure and unavailability. A soft-state registration protocol (can be implemented through either LDAP [HS97], Classified Advertisements [RLS98] or XML) means that state information established due to a notification can be discarded in the absence of continuous subsequent notifications. Thus, the manager can easily cope with failure. Second, the protocol provides a simple way for workstations to un-register from the system providing users with an ease-of-use and sense of autonomy. Machines can simply choose not to send notification messages and no explicit method is necessary for excusing oneself from the system.

4.2. A Note on Morsels

A morsel is storage block of fixed size and is the unit of contribution. The user as such is unaware of its existence and it is an internal storage representation. One can also perceive morsels as containers for the data. The morsel size is of some significance in our system. A small block size would mean a lot of morsels per dataset and correspondingly more bookkeeping and overhead in terms of retrieving, replicating and relocating; while a large block size means lesser morsels

per dataset and consequently low overhead in terms of access, but with high probability of internal fragmentation (discussed in detail below). Thus, careful consideration is required in choosing morsel size requiring it to be flexible and dynamic depending on usage. Our target is large datasets and thus a large morsel size (at least 100 MB) would be in order.

4.3. Scavenge Manager Group and Functionality

In this section, we present the basic functionality of the manager and highlight some issues involved in its realization.

Configuration: Instead of having a single, central manager handling all registration, metadata management and individual scavenger workstations—as in Condor [LLM88] or several other current Grid implementations (Globus MDS [FK98])—we employ a group of workstations to collectively take on the responsibility of managing the system. This group handles scavenger registrations, Grid client requests and all other management activities in a deterministic, load-balanced fashion. Although the central manager approach will drastically simplify our model—and will perhaps suffice for all practical purposes—it is prone to obvious scalability concerns. On the other hand, while the group-based approach provides scalability, it brings with it a range of issues regarding consistency maintenance and Byzantine failure.

All requests are sent to the group which implements a reliable consensus-based group membership protocol to make consistent decisions. For instance, all workstations in the aggregated commodity storage pool see a consistent view of the manager group with a leader to whom all messages are sent. The group conducts an election upon failure of the leader. The leader reliably broadcasts the message among the group members to perform consensus-based decisions—be it request for a file, user workstation withdrawal, etc. Further, all associated metadata maintained by the group will need to be updated consistently. This is an obvious tradeoff between scalability and performance.

Registration Management: The manager collects registration and "Morsel" contribution information from the scavengers and collates it to publish collective storage availability to the Grid. Based on periodic notifications—or lack thereof—from scavengers, the manager can keep track of all available morsels and take appropriate measures in case of disappearing morsels (due to workstation crash, user withdrawal or space reclaim). Measures include updating available storage, ensuring availability of morsel contents by way of additional replication (finding other candidate workstations) and updating metadata. We discuss, in detail, below some of these issues.

File Management: Grid clients request the manager to store bulk datasets as part of their replication process based on the manager's advertisement of its available storage. In response, the manager stores the datasets piece-by-piece using all available workstations enabling parallel access (Figure 2). To implement the piece-by-piece approach, the manager breaks down the file in terms of morsels, finds suitable locations for each morsel and delegates control and maintenance of the morsel to the individual workstation. To start with, locating suitable workstations for morsels can be done in a random fashion based on scavenger registration information held by the manager group. Subsequently, once each scavenger workstation builds some "reputation" of service history—which can be maintained as metadata—the manager can make more informed decisions regarding morsel placement and retrieval. The piece-by-piece model requires bookkeeping in terms of the locations of the various morsels. Further, since the various pieces can reside in several workstations there is the likelihood of more management overhead due to failure or space reclaim in anyone of the individual workstations.

Metadata Management: The manager group maintains the following kinds of metadata. First, we need to maintain registration information indicating workstations' space availability so it can be efficiently regulated, proportioned and published to the Grid. This information is obtained through periodic scavenger updates to the manager. Second, there is the need to maintain directory metadata concerning the datasets stored by the Grid in our scavenging system by providing a namespace for identifying, accessing and retrieving them. Third, information regarding the mapping between datasets and morsels stored in the individual workstations is required for retrieving the dataset. Fourth, a system built on commodity storage almost always needs to address availability with some form of redundancy in place. Finally, with our desire to support Grid based storage management operations—discussed in detail subsequently—we are faced with the need to maintain information on reservation and pinning details.

4.4. Scavenger

The scavenger coordinates the workstations' contribution and involvement in the aggregation environment and its responsibilities can be classified as follows. First, the scavenger describes workstation characteristics and its contribution to the manager using a specification language—Classified Advertisements or XML for instance. Second, all operations—after global decision making at the manager—eventually trickle down to the scavenger which is ultimately responsible for its execution. These include, creating/deleting files, moving data to other scavengers and Grid clients, responding to space reclaim by the user, managing morsels, monitoring the usage of morsels to aid in eviction policies, reserving space, etc. This is by no means a comprehensive listing of scavenger functionality, but only representative of the kinds of operations desired. Third, the scavenger orchestrates a graceful exit in case of user withdrawal from the

system while also ensuring the availability of workstation data.

4.5. Space Reclaim, Relocation and Eviction Policies

One of our key design goals is to revere user autonomy, control and his ultimate authority to withdraw or reclaim space in part or in its entirety (say, due to excessive I/O load, network bandwidth consumption, etc.). One way to address autonomy and non-invasiveness is to have the user allocate a certain amount of space as contribution to the scavenging system and the scavenger performs strictly within those confines—*Passive Scavenging*. This is the approach followed by several flavors of currently available p2p systems. Such a system is relatively easy to construct and the user has a definite sense of control. Space reclaim in such a system can occur due to one of the following reasons.

- Storage morsels have to be relinquished in response to user application needs.
- User expresses a desire to withdraw from the system.
- Manager performs garbage collection to release morsels corresponding to deleted files.

The manager responds to individual scavenger distress signals indicating space shrinkage or user's desire to exit the system by performing several global data management operations followed by data movement based on certain eviction policies. The manager decides which morsels to move where in an educated fashion. One way to approach this is to randomly select morsels and their destinations. In practice, however, this does not always result in optimal utilization of available resources. Alternatively, we could factor in attributes such as morsel usage and workstation characteristics in deciding which morsels to select/evict and where to relocate them. Some strategies for morsels to relocate include the following.

- While attempting to create space in a scavenger workstation in response to space reclaim, we could decide to move the least recently used set of morsels to a workstation determined by the manager.
- Alternatively, if space reclaim occurred often in a particular workstation, we could decide to relocate most recently used morsels elsewhere to improve access rates and avoid overhead due to constant relocation.
- If a subset of morsels is heavily used contributing significantly to the load on the workstation, part of it could be relocated to ensure load balancing.
- Relocate morsels that currently do not satisfy the minimum required replication count (discussed in detail below).
- Relocate morsels corresponding to active files and not deleted ones.
- Relocate morsels serving current client requests to ensure continuity in client access.

Strategies for deciding a destination include the following.

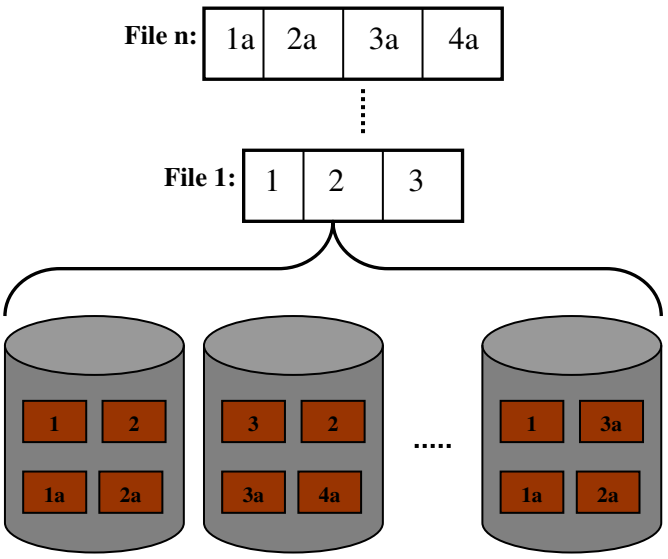

Figure 2: File/Morsel management using the piece-by-piece strategy and availability through morsel replication.

- Perform relocation in a load balanced fashion ensuring that no one scavenger workstation is under utilized or over burdened.
- Minimize the number of recent creations in a scavenger workstation as creations are often followed by bulk data movement.
- Relocate to a workstation with good service and low reclaim or failure history.

Above selection metrics involve significant bookkeeping—at both scavengers and managers—in terms of maintaining "current" metadata regarding not just files, and morsels but their usage access patterns, updating metadata and synchronizing it. Thus, using aggregate data the manager makes decisions on a global scale while relegating local decisions to the scavenger. For instance, the scavenger uses local information to decide which files to evict or which ones need further replication while the manager decides where to relocate them.

4.6. Availability

We address the quality of commodity storage by exploiting the abundance in quantity by eagerly and aggressively replicating morsels of datasets across numerous scavenger workstations (Figure 2). With this approach, anyone particular file has pieces of it replicated across multiple workstations ensuring high availability through redundancy. Such aggressive replication is necessary to handle workstation failure, delayed notifications, transient inaccessibility of machines due to network partitions, user withdrawal or space shrinkage. Since our datasets are write-once-read-many, we need not concern ourselves with consistency and synchronization issues. In our system, we are concerned about the availability of two kinds

of data namely, manager metadata and file data. Below are a few replication strategies.

- Metadata is proactively replicated and synchronized among manager group members to ensure consistent global decision making. The manager group is at the most a handful of machines and thus consistency of metadata can be maintained and tamed.
- A minimum replication factor for each file is maintained to ensure availability—i.e., each morsel in a file is at the very least replicated "so many times" across different workstations.
- When a file is initially created by the manager, it follows some of the replication policies outlined in Section 3.2.5 to decide upon destinations. The manager also maintains metadata concerning the morsel replica locations.
- Once minimum replication factor is guaranteed, morsels can be replicated further depending on temporal access patterns and popularity.
- With replication comes the question of which replica to choose for any given morsel. The manager metadata maintains—from periodic updates from scavengers over a period of time—morsel characteristics which include access rates, load, service history and popularity (as a measure of clients requesting data from that particular workstation). A function of some of the aforementioned attributes can be used to decide which workstation to access the morsel from.

4.7. Grid Awareness

Our solution is intended for use as a Grid storage fabric to store and access large datasets at high sustained rates. To achieve this goal we address the following features.

Information: The ability to discover properties about the scavenging system in the context of a Grid Information Service is highly desired. Apart from the local information management at the scavenger workstations and the managers, we would require support for Grid information providers collating information about the several scavengers, storage space available, transfer protocols supported, etc., and notifying state information to directory services (say, Globus MDS).

Protocol Agnostic: The Grid is replete with several high-speed bulk transfer protocols with varied benefits (GridFTP, SRB, IBP, NFS and HTTP). Thus, our local storage solution should be agnostic enough to support transfers through anyone of the aforementioned tools. NeST provides similar protocol agnostic behavior but for a standard file system based storage. Our position is complicated due to the induction of numerous individual workstations. Our approach is to expose the transfer protocols supported by the aggregate commodity storage to the information service through local information providers so they can be discovered by clients. Clients can then choose the appropriate protocol to initiate the transfer. Locally at the aggregate site, however, we will need a virtualization layer so these requests can be translated to the individual workstations.

Distributed Space Reservation and Pinning: Grid data transfers are usually preceded by requests for space reservations followed by bulk transfers. Tools such as GARA [FK98] and NeST or SRM provide middleware for co-allocated reservations across multiple domains and reservations on single local storage system respectively. The challenge for us is to enable, support and guarantee distributed space reservation operations across numerous commodity scavenger workstations and several types of space guarantees required by the Grid (volatile, persistent).

Yet another commonly required Grid operation is the ability to "pin" datasets to their locations (workstations) which guarantees their availability at a future point in time. SRM supports several pinning strategies for a hierarchical storage environment. Our challenge is to translate such mechanisms to the scavenging environment which poses new questions. We need to address pinning multiple morsels belonging to a file that may reside at different workstations, maintain pin integrity in the face of constant data relocation due to space reclaim, etc.

Both distributed pinning and space reservations warrant the need for a two-phase commit protocol. The manager—after choosing specific morsel locations in response to a file fetch—locks down the morsels on the individual scavenger workstations so they do not disappear due to user withdrawal, or space shrinkage. These "locks" are later translated down from the manager to the individual scavenger so the morsels are available for access. Similar strategies are adopted for space reservations so morsels can be placed initially. Once all corresponding scavengers answer in the affirmative for space availability for storing morsels or morsel availability for access, the manager proceeds ahead with either storing the morsels or fetching them to/from the scavenger workstations.

Security: In terms of Grid security, there are questions regarding clients authenticating to the manager and individual workstations. We could, for instance, draw analogies from Condor-G [CONDOR04] that accepts Grid client requests to submit jobs to a Condor pool. This is similar to our case of Grid clients requesting data from the scavenging pool. Once authenticated by the manager, clients should be able to transparently fetch morsels from any scavenger workstation if necessary. In terms of data integrity itself, morsel encryptions and checksums would be in order.

Transparent Access: The system should provide transparent interface to clients in terms of data discovery, access and transport. Clients should be able to access files stored using standard file system syntax often provided by typical high-end storage systems. We address this by providing a namespace as

an external visible interface for all the datasets stored, while internally maintaining associations between namespace entries, scavenger workstations and morsels. A namespace also provides the manager a convenient way to organize numerous workstations, hundreds and thousands of files and multitudes of morsels.

5. Related Work

At this time we review related work in distributed file systems research and point out differences from our effort.

Networked and Distributed File Systems: Tens of networked and distributed file systems have been built since the eighties addressing issues such as performance (NFS [Nowicki89], CIFS [LP96] and PODOS [VSM02] file system), transparency (LOCUS [PW85]) and availability (AFS [AFS95] and CODA [CODA87]). These approaches either use centralized servers (like in NFS) or distributed replicated file servers (as in CODA or Zebra [HO93]) to support numerous clients demanding remote file access. Such techniques, while apt for their respective design choices, still require high maintenance in terms of administering the file servers.

Parallel File Systems: A parallel file system offers a high-performance alternative to distributed file systems, comprising of a set of computers providing uniform name space to a cluster. It provides high-performance, high-availability and reliability but requires high-speed communication capabilities and does not translate well to loosely connected environments. Such systems are popular in the supercomputing and Grid environments (PVFS [CLR+00], Lustre [LUSTRE01], Petal [LT96], GPFS [SH02] and Shark [HS96] or XFS [SDH+96]).

Serverless File Systems: Serverless file systems address performance and scalability by removing the server bottleneck in the aforementioned systems. Prominent examples in this category are xFS [WA93], GFS [GGL03] and FARSITE [ABC+02] that attempt to build a unified file system name space using a network of workstations in a loosely connected environment. They are similar to our proposed approach in addressing loosely connected workstations and serverless behavior but differ in their lack of local autonomy, desire to provide unified name space (very costly in a networked workstation environment) and concentration on typical file usage patterns (with the exception of GFS which addresses data intensive needs).

Peer-to-Peer Storage: Peer-to-peer [CP02] storage techniques thrive on the premise that individual workstations contribute storage to the pool in return for access from the pool. Several Internet-scale p2p storage systems—PAST [DR01], OceanStore [KBC+00], CFS [DKK+01]—are being constructed to provide a persistent, scalable, highly available, decentralized storage infrastructure. In theory, such a system could perhaps be used to harness idle storage in workstations within a domain for use by the Grid. In practice however, much of p2p storage research is motivated by design choices concerned with wide-area environments as opposed to a corporate LAN setting. With our design setting and assumptions involving sharing within a domain, we can optimize in terms of performance, administration, security and availability. Further, Grid access to bulk data requires specialized storage management operations and sustained delivery performance which p2p storage techniques do not provide.

Grid Storage Services: Systems such as LegionFS [WWH+01], SRM [SSG02] and IBP [PBE+99] provide both local storage management and Grid scheduling middleware. For instance, LegionFS provides a location transparent wide-area file system using an object based system; SRM is a storage resource manager—similar to compute resource manager—providing middleware components for space and file management on storage resources for the Grid; IBP is an Internet scale middleware designed to provide a global shared storage service, implemented as part of the network fabric. Just as IP is an abstraction based on data-link layer, IBP is an abstraction based on blocks of disk data. Much as how a peer-to-peer storage system can be used to build a local scavenging environment, IBP depots can also be used in a similar context. However, given the design of both p2p and IBP-like systems, each node would behave as a router which can be quite expensive in a local scavenging realm. SRB [RWM02] and GASS [FK98] both provide middleware for uniform access to heterogeneous storage resources in the Grid environment. On the other hand, NeST [BVL+02] is strictly a local storage management solution that can be integrated into many Grid scheduling middleware. NeST offers storage management operations such as lot management, reservations, etc., and can be used with many Grid data movement protocols (GridFTP [FK98] and Chirp [CONDOR04]).

Several of the aforementioned solutions are for a widely distributed Data Grid setting that use storage clusters or high-end storage. In contrast, our aggregate commodity storage proposition is intended to be a local storage management solution which can be integrated with Grid middleware systems such as GASS and SRB or can be plugged into NeST as well.

6. Conclusions

In this paper, we have presented our initial design on the construction of a distributed storage infrastructure through the aggregation of commodity user workstations and scavenging space from them. Some of our guiding principles have been to address the “quality” of workstations by aggressive replication and encryption; exploit their “quantity” to aggregate staggering storage; revere user autonomy through space reclaims and morsel evictions; provide scalability through soft-state registration, separation of concerns between

scavengers and the manager group, etc.; and provide a transparent interface for Grid clients. Subsequent to the realization of the architecture described in this paper, more sophisticated strategies and issues—including performance optimizations through I/O bandwidth aggregation, proactive scavenging of space to emulate CPU cycle stealing and constructing a hierarchy of storage services between high-speed, aggregated and archival storage based on data aging—can be addressed and built atop.

References

[ABC+02] A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, R. P. Wattenhofer, *FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment, in Proc. 5th OSDI*, Dec 2002.

[AFS95] *AFS Reference Page*, http://www-2.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/afs.html, 1995.

[BVL+02] J. Bent, V. Venkataramani, N. Leroy, A. Roy, J. Stanley, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny, Flexibility, Manageability, *and Performance in a Grid Storage Appliance*, *in Proc. of IEEE HPDC-11*, July 2002.

[CLR+00] P. Carns, W. Ligon III, R. Ross, and R. Thakur, *PVFS: A Parallel File System For Linux Clusters, in Proc. of the 4th Annual Linux Showcase and Conference*, pp 317-327, 2000.

[CODA87] CODA File System, http://www.coda.cs.cmu.edu/, 1987.

[CONDOR04] *Condor, http://www.cs.wisc.edu/condor,* 2004.

[CP02] J. Crowcroft, and I. Pratt. *Peer to Peer: peering into the future*. in *Networks 2002*. 2002.

[CW95] R.A. Coyne and R.W. Watson. The Parallel I/O Architecture of the High-Performance Storage System (HPSS). In *IEEE MSS Symposiu*m. IEEE Computer Society Press, 1995.

[DataGrid02] *The Data Grid Project, http://www.eu-datagrid.org, 2002*.

[DKK+01] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, *Wide-area cooperative storage with CFS, In Proc. 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[DPSS03] *Distributed Parallel Storage System, http://www-didc.lbl.gov/DPSS/,* 2003.

[DR01] P. Druschel and A. Rowstron, *PAST: A large-scale, persistent peer-to-peer storage utility, in Proc. of HOTOS Conf.,* 2001.

[DSG03] *DOE Science Grid, http://www.doesciencegrid.org/,* 2003.

[ESG03] *Earth System Grid, https://www.earthsystemgrid.org/,* 2003.

[FK98] I. Foster and C. Kesselman. *The Globus Project: A Status Report*. in *IPPS/SPDP '98 Heterogeneous Computing Workshop*. 1998.

[GGL03] S. Ghemawat, H. Gobioff, S. Leung, *The Google file system, in Proc SOSP 2003.*

[GriPhyN02] *The GriPhyN Project, http://www.griphyn.org, 2002*.

[HJS+00] W. Hoschek, J. Jaen-Martinez, A. Samar, and H. Stockinger. *Data Management in an International Grid Project*. in *2000 Internationsl Workshop on Grid Computing (GRID 2000)*. 2000. Bangalore, India.

[HO93] J. Hartman and J. Ousterhout. *The Zebra Striped Network File System, Proc. 14th Symposium on Operating Systems Principles,* pp. 29-43, December 1993.

[Holtman00] K. Holtman, *Object Level Replication for Physics*. in *4th Annual Globus Retreat*. 2000. Pittsburgh.

[HS96] R. Haskin and F. Schmuck. *The tiger shark file system. In Proc. IEEE Computer Conference,* 1996.

[HS97] T.A. Howes and M.C. Smith. *LDAP Programming Directory Enabled Application with Lightweight Directory Access Protoco*l. Technology Series. MacMillan, 1997.

[HSS00] M. Hafeez, A. Samar, and H. Stockinger. *Prototype for Distributed Data Production in CMS*. in *7th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT2000)*. 2000.

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C Wells, B. Zhao, *OceanStore: An Architecture for Global-Scale Persistent Storage. In Proc. ASPLOS,* December 2000.

[Kerberos03] *Kerberos: The Network Authentication Protocol, http://web.mit.edu/kerberos/www/,* 2003.

[LEGION03] *Legion: Worldwide Virtual Computer, http://www.cs.virginia.edu/~legion/*, 2003.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor – A Hunter of Idle Workstations. In *Proc. 8th Intl Conf. on Distributed Computing System*s, pages 104–111, 1988.

[LSZ+02] H. Lamehamedi, B. Szymanski, S. Zujun, and E. Deelman, *Data Replication Strategies in Grid Environments*, *in 5th International Conference on Algorithms and Architecture for Parallel Processing, ICA3PP'2002*, Bejing, China, pp. 378-383, October 2002.

[LIGO02] *The LIGO Experiment, http://www.ligo.caltech.edu/, 2002*.

[LP96] P. Leach and D. Perry, *CIFS: A Common Internet File System, in Microsoft Interactive Developer*, November 1996.

[LT96] E.K. Lee and C.A. Thekkath. *Petal: Distributed Virtual Disk,. Proc. ASPLOS-7*, October 1996.

[LUSTRE01] *Lustre Technical Project Summary*, *Technical Report*, Cluster File Systems, Intel Labs, June 2001.

[Merkey94] P. Merkey, *Beowulf Project at CESDIS*, http://beowulf.gsfc.nasa.gov/, 1994.

[MMR+01] D. Malon, E. May, S. Resconi, J. Shank, A. Vaniachine, T. Wenaus, and S. Youssef. *Grid-enabled Data Access in the ATLAS Athena Framework*. in *Computing and High Energy Physics 2001 (CHEP'01) Conference*. 2001.

[NM02] H. Newman and R. Mount. *The Particle Physics Data Grid, www.cacr.caltech.edu/ppdg*.

[Nowicki89] Nowicki, B., *NFS: Network File System Protocol Specification*, *Network Working Group RFC1094,* March 1989.

[PBE+99] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. *The Internet Backplane Protocol: Storage in the network. In Proc. NetStore '99: Network Storage Symposium. Internet2,* October 1999.

[PW85] G.J. Popek and B. J. Walker, eds., *The LOCUS Distributed System Architecture*, MIT Press, Cambridge, MA, 1985.

[RLS98] R. Raman, M. Livny, and M. Solomon, *Matchmaking: Distributed Resource Management for High Throughput Computing, In Proc. 7th IEEE Symp. on High Performance Distributed Computin*g. IEEE Computer Society Press, 1998.

[RWM02] A. Rajasekar, M. Wan, and R. Moore, *MySRB & SRB – Components of a Data Grid*, *In Proc. 11 th International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, 2002.

[SDH+96] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, , and G. Peck, *Scalability in the XFS file system. In Proc. USENIX Technical Conference,* pp. 1-14, January 1996.

[SETI03] *SETI@Home, http://setiathome.ssl.berkeley.edu/,* 2003.

[SH02] F. Schmuck and R. Haskin, *GPFS: A Shared-Disk File System for Large Computing Clusters, In Proc. of the First Conference on File and Storage Technologies (FAST)*, Jan. 2002.

[Sprite96] The Sprite Operating System, http://www.cs.berkeley.edu/projects/sprite/sprite.html, 1996.

[SSG02] A. Shoshani, A. Sim, and J. Gu. *Storage Resource Managers: Middleware Components for Grid Storage. In Proc. Nineteenth IEEE Symposium on Mass Storage Systems (MSS '02)*, 2002.

[TBS+01] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny, *The Kangaroo approach to data movement on the grid, In Proc. of the Tenth IEEE Symposium on High Performance Distributed Computing,* San Francisco, California, August 2001.

[TeraGrid04] *TeraGrid: http://www.teragrid.org,* 2004.

[TWD95] D. Teaff, D. Watson, and B. Coyne, *The Architecture of the High Performance Storage System (HPSS), in Proc. of the Goddard Conference on Mass Storage & Technologies,* College Park, Maryland, March 1995.

[VSM02] S. Vazhkudai, J.M. Syed, P.T. Maginnis, *PODOS: The Design and Implementation of a Performance Oriented Linux Cluster, in Proc. Journal of Future Generation Computer Systems - Special Issue on Cluster Computing*, Volume 18, Issue 3, pp. 335-352, January 2002.

[WA93] R. Wang and T. Anderson. *xFS:A Wide Area Mass Storage File System. In Proc. Fourth Workshop on Workstation Operating Systems,* pp. 71-78, October 1993.

[WWH+01] B. S. White, M. Walker, M. Humphrey, and A. S. Grimshaw. *LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications, In Proc. of Supercomputing'01*, 2001.