File Grouping for Scientific Data Management: Lessons from Experimenting with Real Traces

Shyamala Doraimani
Computer Science and Engineering
University of South Florida
Tampa, Florida, USA
sdoraimani@gmail.com

Adriana Iamnitchi
Computer Science and Engineering
University of South Florida
Tampa, Florida, USA
anda@cse.usf.edu

ABSTRACT

The analysis of data usage in a large set of real traces from a high-energy physics collaboration revealed the existence of an emergent grouping of files that we coined "filecules". This paper presents the benefits of using this file grouping for prestaging data and compares it with previously proposed file grouping techniques along a range of performance metrics. Our experiments with real workloads demonstrate that filecule grouping is a reliable and useful abstraction for data management in science Grids; that preserving time locality for data prestaging is highly recommended; that job reordering with respect to data availability has significant impact on throughput; and finally, that a relatively short history of traces is a good predictor for filecule grouping.

Our experimental results provide lessons for workload modeling and suggest design guidelines for data management in data-intensive resource-sharing environments.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*distributed systems*

General Terms

Experimentation, Measurement, Performance.

Keywords

File grouping, science grids, trace analysis, data management, caching, job scheduling.

1. INTRODUCTION

The importance of data staging for data-intensive collaborations in well accepted: much of today's science is supported by international collaborations among tens or hundreds geographically remote institutions that produce, analyze, and exchange Terabytes of data per day. Staging data locally is thus paramount for efficient access to data.

HPDC'08, June 23–27, 2008, Boston, Massachusetts, USA.

It is well acknowledged that scientific applications often process multiple input files [3, 36], yet, few studies [23, 16] provide a quantitative characterization of this pattern in scientific workloads. In addition, efforts to provide benchmarks for Grid workloads [27, 29, 28] have been challenging due to the scarcity of available traces from mature Grid communities, the diversity of Grid applications, and the heterogeneity of software middleware platforms currently deployed.

This lack of evidence in usage characteristics has several significant outcomes. First, resource management solutions (such as data prefetching, job and data-transfer scheduling) are designed for and evaluated on synthetic workloads whose accuracy of modeling the real world is difficult to judge. Second, quantitative comparison of different solutions to the same problem becomes impractical due to different experimental assumptions and independently generated synthetic workloads. And third, solutions are designed in isolation, to fit the particular and possibly transitory needs of specific groups.

In an attempt to alleviate this problem, we use real traces from a scientific data-intensive community and try to infer lessons that can be applied to data management in grids in different domains with data-intensive activities. This effort is part of a more general research approach that aims to exploit patterns in real user traces to design better techniques for resource management [26, 24].

This paper presents our experiments with various staging techniques for groups of files using real traces from a production-mode data-intensive high-energy physics collaboration, the DZero Experiment [1], hosted at Fermi National Accelerator Laboratory (FermiLab).

The contributions of this paper are:

- A set of recommendations for workload modeling for data-management in science grids based on our experiments with real traces. We acknowledge that one set of real traces cannot provide a workload model general enough to fit other domains and diverse applications. However, it is enough to prove that generalizing previously accepted patterns (such as file size and popularity distribution) from systems such as the Web or filesystems is not appropriate.

- A set of design recommendations for data staging for distributed collaborations, such as preserving time locality and enforcing the role of job reordering.

- Our previous analysis of the DZero traces showed the existence of *filecules* [fil'-eh-kyul'], that are groups of files always used together in data processing jobs. We show that a simple and well-understood caching algorithm (i.e., least recently used) that uses filecules has significant advantages over more sophisticated caching techniques. We identify the

property of the traces that leads to this performance difference and make a set of recommendations for accurate workload modeling.

- An implementation and comparison of previously proposed caching mechanisms to quantify the relative benefit of each solution on real traces. We took the trouble of implementing previously proposed solutions to better understand them and accurately compare them with other approaches on the set of real traces available for our study.

This paper is structured as follows. Section 2 gives an overview of the file grouping identified in and briefly presents the characteristics of the DZero traces. Section 3 describes the simulator we built and the set of data management algorithms implemented. Experimental results are presented in Section 4 as a set of lessons inferred from our study. Section 5 reviews related work on file grouping and techniques for data management that consider file groupings. Finally, Section 6 summarizes our results and describes future work.

2. FILECULES IN DZERO

This study is motivated by the identification of emergent file groups in the DZero traces [23, 16].

Filecules [fil'-eh-kyuls'] were introduced in [23] as disjoint groups of files characterized by simultaneity of access. Inspired from the definition of molecules, we defined a filecule as an aggregate of one or more files in a definite arrangement held together by special forces related to their usage. We thus consider a filecule as the smallest unit of data that still retains its usage properties. We allow one-file filecules as the equivalent of a monatomic molecule, (i.e., a single-atom as found in noble gases) in order to maintain a single unit of data (instead of multiple-file filecules and single files).

Formally, a set of files $f_1, ..., f_n$ form a filecule F if and only if $\forall f_i, f_j \in F$ and $\forall F'$ such that $f_i \in F'$, then $f_j \in F'$. Properties that result directly from this definition are:

1. Any two filecules are disjoint.
2. A filecule has at least one file.
3. The number of requests for a file is identical with the number of requests for the filecule that includes that file.
4. The lifetime of filecules is the same as of the files it includes. Moreover, all files in a filecule have the same lifetime. In this article, we define the lifetime of a file as the interval between the first and the last request for that file as seen in our workloads.

The identification of filecules does not require a priori knowledge about the type of data processed. Instead, filecules are identified at run time using historical information about simultaneous access to multiple files. The remainder of this section briefly presents the DZero workloads used in our experiments and the main filecule characteristics with direct impact on data staging.

2.1 The DZero Workload

The DZero Experiment [1] is a virtual organization consisting of hundreds of physicists in 70+ institutions from 18 countries. It provides a worldwide system of sharable computing and storage resources used for extracting physics results from several Petabytes of measured and simulated data [34, 51].

The studies presented in this research use file access traces recorded between January 2003 and March 2005. This workload, referred in this article as "DZero workload", contains information about the jobs submitted by the DZero community and the data files used as input for these jobs. The workload consists of detailed access information to 996,227 files processed by 113,062 jobs submitted by 499 users over the 27 months of the workload. File information includes the files that have been requested with every job and their size. Detailed analysis of the workload is described in [16].

2.2 Filecule Characteristics

File groupings such as filecules can be efficiently used for data staging or caching. Since often times jobs need to process all input files at once, storing files that are likely to be used together at the same storage location may improve job turnout or guide job scheduling. A set of data and access characteristics (such as size and popularity) affect significantly the performance of resource management algorithms. Various workload models have been proposed, typically generalizing models from other systems: for example, the Zipf distribution of web page access has been considered the de facto model for data access distribution. Similarly, the log-normal file size distribution inferred from evaluation of different file systems (Windows and Unix) seems a intuitive model to be used in synthetic workload generators for the Grid.

Our analysis of the DZero traces infirms this intuition. In particular, we discover that:

- at the scale of scientific data files, file sizes are artificially limited to the maximum file size supported by the filesystem. This fact distorts the file size distribution in unexpected ways.
- the science community exhibits a different file popularity distribution than the Web. In particular, the interest in files seems more uniformly distributed.

Table 1 presents statistics on size, popularity and lifetime for files and filecules.

2.2.1 Size Distribution

Figure 1 shows the distribution of the number of files per filecule: about 56% are one-file filecules.

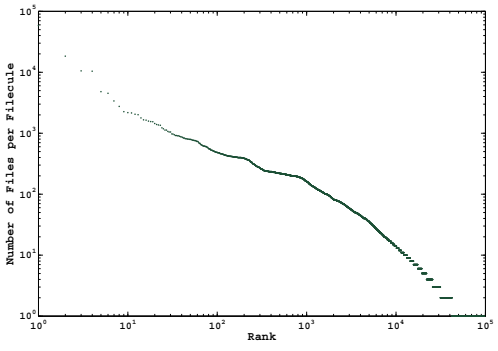

Figure 1: Number of files per filecule

Figure 3 shows the filecule size distribution, ranging from 15.7 TB (or 18,326 files) for the largest filecule to 23.5 KB for the smallest. About 5% of the filecules are above 15 TB. All the 2,264 files of size less than 23.5 KB have been grouped into filecules. The largest filecule size is about 8,000 times the largest file size.

Log-logistic distribution with parameters mentioned in Figure 3 best fits filecule size distribution. This contradicts the log-normal

Table 1: Statistics of size, popularity and lifetime for files and filecules

Property	Minimum	Maximum	Mean	Median	Standard Deviation
File size	234 bytes	1.98 GB	0.3859 GB	0.3773 GB	0.3230 GB
Filecule size	23 KB	16,051 GB	3.9859 GB	0.9419 GB	54.5137 GB
File popularity	1	996	12	3	25
Filecule popularity	1	996	41	30	50
File lifetime	15 secs	27 months	4 months	1 month	5 months
Filecule lifetime	15 secs	27 months	8 months	7 months	5 months

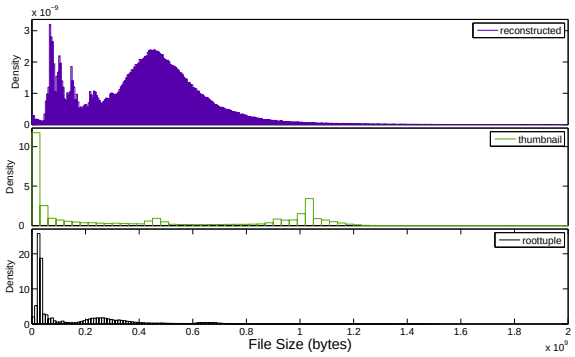

Figure 2: File size distribution per data tier. Each data tier contains files generated as a result of various types of data processing.

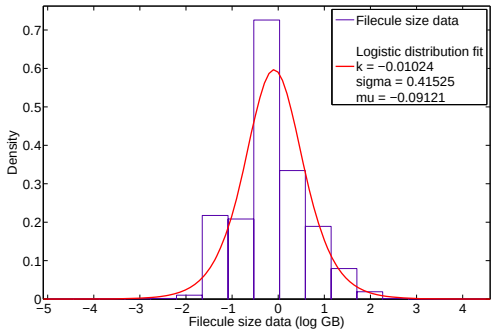

Figure 3: Filecule size distribution

size distribution of data observed in [17] and [12]. The difference between log-normal and log-logistic distribution is that the log-logistic has a fat tail (larger number of large files). The curve of a log-logistic distribution increases geometrically with small values, flattens in the middle and decreases slowly at high values.

2.2.2 Popularity Characteristics

Popularity of file or filecule is measured as the number of times a file or filecule has been requested. and is particularly relevant for predicting caching performance.

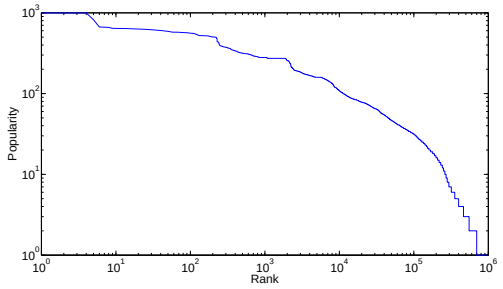

Figure 4: File popularity vs. rank

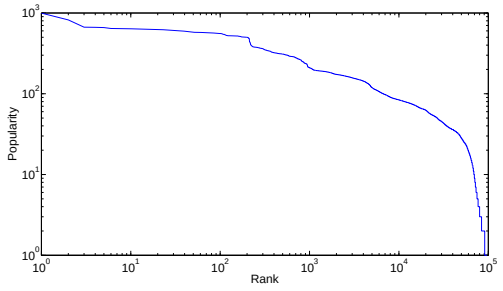

Figure 5: Filecule popularity vs. rank

Figure 4 shows the file popularity distribution. The most popular file was requested by 996 jobs (34 unique users). About 30% of the files have been used by only one job (file popularity =1). The rank for the median file popularity (3 jobs) is 700,000, i.e., about 30% of the total number of files (997,227) are requested only by one or two jobs. 6.5% of the total number of files (65,536 files ordered by file popularity) account for 45% (5,247,549 requests) of total requests (11,568,086 requests). This is similar to the observation in [9] where a small set of files account for majority of the requests. Only 4 files are highly popular with a file popularity of

996. The popularity distribution is heavy tailed (Figure 4) similar to observations in [25] and [7]. Also, the distribution does not follow the Zipf observed in [47], [4], [14] and [7].

Figure 5 shows the filecule popularity. There is only one filecule with maximum popularity observed. The generalized Pareto distribution fits best the filecule popularity data. The filecule popularity is not as heavy tailed as file popularity because files with less popularity group better into filecules than the very popular files. This is similar to the discussion in [5], which mentions that there might be a few more popular files which will be used along with a lot of different file sets. Hence, when trying to identify disjoint groups of files, these files remain single rather than form groups.

2.2.3 Lifetime Characteristics

Defined as the time between the start time of the first job that requested a file and the end time of the last job that accessed it in the workloads analyzed, lifetime suggests how long data is worthwhile storing in cache.

Figure 6 shows the filecule lifetime distribution as an extreme value distribution with shape parameter (k)=–0.1539, scale parameter (sigma)=3,649.1 and location parameter (μ)=4,612. In the Windows filesystem the lifetime was shown to be hyperexponential [17]. More than 70% of the filecules (equivalent to 35% of the files) are active after 5 months, while about 5% (which include 30% of the files!) of filecules become inactive after a day.

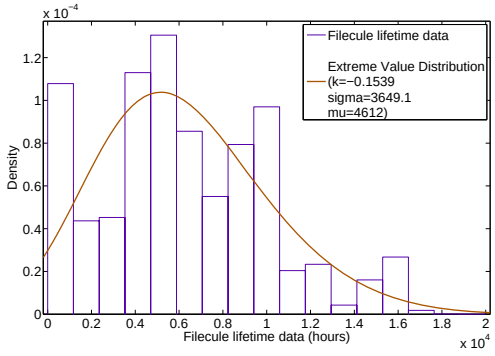

Figure 6: Filecule lifetime distribution

40% (396,341) of files have lifetime shorter than one week. Median file lifetime is 712 hours (≅1 month). About 35% of the files have a lifetime greater than or equal to 5 months. This is similar to the observation in a web proxy workload [7], where 20% of the files were active after 5 months. 294,355 (30%) files have a lifetime of less than or equal to a day (24 hours). This number is less than the ones mentioned in [7] which reports 50% of inactive files on the next day. This indicates that the files in DZero have longer lifetimes than those observed in the Internet. This can influence the effectiveness of caching. On average, 4.54 TB of distinct data is requested each day. 30% of this data becomes useless on the next day and 70% of the data is still active. In order for a cache to be effective, it needs to retain around 70% of the data from the previous day. This shows that any cache size that is less than 3.5 TB (70% of 5 TB) will not be enough to take advantage of temporal locality.

3. EXPERIMENT DESIGN

The main objective of our experiments is to understand the impact of real traces on data staging for data-intensive scientific processes. Along the way, we are interested in evaluating the potential of filecules for data prefetching and to comparing it with other relevant techniques.

This provides a set of lessons in two important directions. First, our experience with real traces can inform the design of future data management techniques for data-intensive scientific collaborations from different domains. Second, our results suggest a range of realistic and relevant parameters for modeling and generating synthetic workloads adapted to the particularities of a specific application domain.

To this end, we implemented a set of data staging and job scheduling algorithms (presented in Section 3.1) in a simulator described in Section 3.3. The experiments were performed using the DZero traces and evaluated based on the performance metrics presented in Section 3.2.

Table 2: Data management algorithms tested.

Algorithm	Caching	Scheduling	File grouping
File LRU	LRU	FCFS	None
Filecule LRU	LRU	FCFS	Optimal Filecule
GRV	GRV	GRV	GRV
LRU-Bundle	LRU	GRV	None
Filecule-LRU-Bundle	LRU	GRV	1-month Filecules

3.1 Cache Replacement and Job Scheduling Algorithms

In the context of Grid resource management for data-intensive applications, it has been shown that data staging and job scheduling work best together for higher job throughput and lower communication costs. We thus experiment with combinations of two data staging and two job scheduling techniques.

The first data staging algorithm is the well known Least Recently Used (LRU) cache replacement algorithm, a simple and well understood algorithm that is widely used in practice (for example, it is used in DZero). We test LRU in the classical form with files as data granularity and no prefetching. In addition, we add prefetching based on filecule grouping: when a file that is not in the local storage is requested, the filecule to which the file belongs will be prefetched and the least recently used filecules will be evicted to make room, if necessary.

The second staging algorithm is Greedy Request Value (GRV), proposed by Otoo et al. in [37, 39, 38]. In GRV, files are staged based on a *request value* that captures historical usage information such as file popularity, normalized size, and other parameters described below.

The two scheduling algorithms used are First Come First Serve (FCFS) and the one proposed by GRV, where jobs are scheduled in the decreasing order of their request values (details below). In FCFS, jobs are scheduled in the order they are submitted; data prefetching decisions are thus informed by the data needs of the first job in queues.

Two file grouping techniques are compared: filecules and the grouping emergent from the cache membership implied by the GRV prefetching technique. The algorithms implemented and compared in this paper are summarized in Table 2.

The Greedy Request Value (GRV) algorithm combines cache replacement with job scheduling with the purpose of taking advantage of current cache content by reordering jobs such that the jobs with most data already in the cache are preferred. In GRV, each file f_i requested is assigned a relative value v_{f_i} based on its size

$s(f_i)$ and its popularity $n(f_i)$(Equation 1). Each job is assigned a relative value V_r based on the popularity $n(r)$ of its set of input files r and the relative values of the files in r (Equation 2). Jobs are scheduled in a priority scheduling manner with higher priority associated to jobs with higher relative value V_r.

$$v_{f_i} = \frac{s(f_i)}{n(f_i)} \quad (1)$$

$$V_r = \frac{n(r)}{\sum_{i=1}^{N} v_{f_i}} \quad (2)$$

In GRV data is prefetched as needed to support the execution of the jobs with highest priorities. As in all experiments presented here, we assume infinite computational resources: therefore, jobs are only delayed due to storage constraints. Data used by a job is maintained in cache during the entire job execution (job execution times are provided in the DZero traces). If no jobs are in queue, the files with the highest relative values are prefetched.

Therefore, the data staging algorithm in GRV selects the files to be prefetched or maintained in the cache based on their size and popularity information. Filecules, in contrast, respond only to the inter-file relationships as observed from access patterns: size and popularity are ignored, but membership to the same data input is recorded and exploited.

The GRV scheduling component requires the re-evaluations of all request values when a new job enters the queues. Job queue freezing can be used to limit the computational overhead, the risk of job thrashing, and starving: a new job that arrives at the queue waits until all the jobs in the frozen queue have started to run. Queue freezing also takes advantage of the temporal locality characteristic in the workload. We implemented queue freezing in LRU-GRV and GRV with 1000 jobs as the freezing threshold (a somewhat arbitrarily chosen value that approximates the average number of jobs submitted per week in the DZero workload).

We propose a combination of LRU cache replacement with GRV-based job reordering that we refer to LRU-Bundle. A version that adds filecule-based prefetching (Filecule-LRU-Bundle) is also analyzed. We will show that the LRU-Bundle combination is highly efficient, as it takes advantage of workload characteristics (time locality) while making good use of the cache status (via GRV job reordering).

3.2 Performance Metrics

The traditional metric used in evaluating cache replacement algorithms is byte hit rate. However, when prefetching data from remote storage, the volume of data transfered is an important metric for evaluating performance. Other metrics of interest are job waiting time and the computational overhead involved in job reordering.

In our experiments, we measure the following:

(1) Byte hit rate indicates the percentage utilization of the content of the cache.

(2) The percentage of cache change is a measure of the amount of data transferred to the cache in order to run a job. For cache replacement algorithms that do not involve data prefetching, byte hit rate is enough to account for both data transfers and cache utilization. For algorithms that involve data prefetching, both percentage of cache change and byte hit rate needs to be measured. A better cache replacement algorithm has higher byte hit rates and lower percentage of cache change. We measure the percentage of cache change as the percentage of the difference in bytes in cache before and after the cache is updated to accommodate the next job.

(3) Job waiting time indicates how long a job spends in the waiting queue between submission and the time it has all the data available on the local storage. We ignore the data transfer time both for simplicity and to better isolate the performance of the algorithms we evaluate. Because we assume infinite computing resources, the job waiting time is only affected by job reordering and storage constraints.

(4) Scheduling overhead can be represented as the number of computations performed to make a scheduling decision. FCFS has no scheduling overhead. GRV-based scheduling overhead depends on the number of jobs in the waiting queue and the number of input files of each job.

3.3 Simulator Implementation

Our experiments are simulations of disk caches using real workloads from the DZero Experiment. In all cases, we consider infinite computation resources, which allows us to isolate the effects of storage constraints from those of CPU constraints. This assumption is also realistic for many data-intensive scenarios, where data management component is the bottleneck. We implemented the algorithms listed in Section 3.1 and compared them using the metrics listed in Section 3.2.

The DZero workload is accessed from a MySql database. The workload consists of two sets of information: information about the start time and end time of each job and information about the list of files requested by each job. The simulator is a Java program consisting of 3,500 lines of code which connects to the MySql database using JDBC. The input values to the simulator are the size of the cache, the cache replacement algorithm and the job scheduling algorithm. When not specified otherwise, the simulations were run on the entire workload.

A job run and, in turn, data transfers can be triggered by two events: a job arrival or a job completion. When the total size of data requested by a job exceeds the cache size, the job is ignored.

Simulations were run for 6 different caching algorithms with 5 different cache sizes leading to 30 different runs. For calculating the optimal set of files to be loaded into the cache for the GRV algorithm, the history of jobs from the previous 1 week was used. The average run time for each simulation is around 6 hours on a Pentium II Linux machine with 2 GB memory.

4. LESSONS FROM EXPERIMENTAL RESULTS

We structure our presentation and discussion of experimental results along the lessons learned from this effort. Along the way, we will present the performance of the algorithms tested on the DZero traces in terms of byte hit rate, percentage of cache change, computational overhead, and job waiting time.

4.1 Lesson 1: Time Locality

The filecule definition preserves the time locality of file accesses because the files in a filecule (by definition) are requested by the same data processing jobs. Other file grouping techniques may give more weight to other criteria, such as the relative size of a file and its popularity. What is the effect of such alternatives?

In the following we compare two data management metrics relevant when prefetching groups of files. The first is the traditional byte hit rate and shows the benefits of data staging. The other is percentage of cache change and quantifies the staging costs in terms of data transfer.

Figure 7 presents the byte hit rates of all the algorithms for different cache sizes. Filecule LRU has the highest byte hit rate for

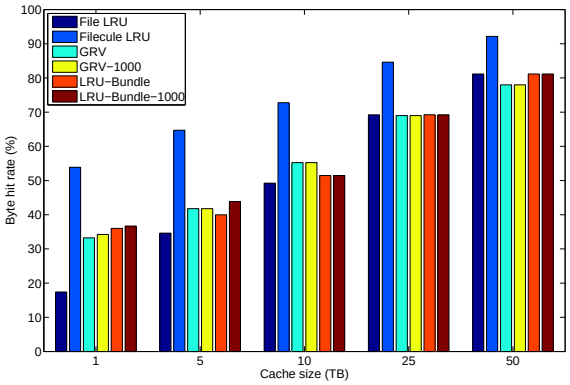

Figure 7: Average byte hit rate. Cache sizes correspond, in increasing order, to 0.3%, 1.3%, 2.6%, 6% and 13%, respectively, of the total amount of data accessed.

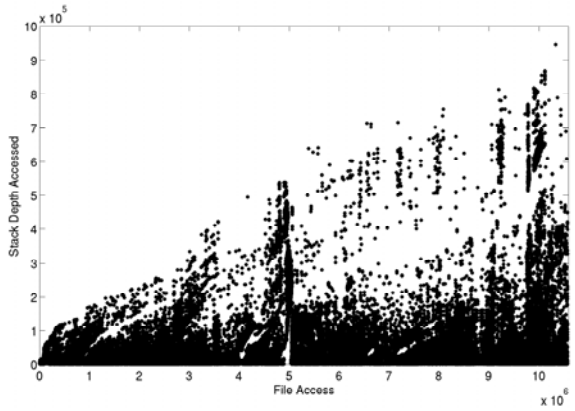

Figure 8: Stack depth analysis of file requests.

Table 3: Stack depth analysis: statistics

Measure	Value
Maximum	946,600
1 percentile	85
10 percentile	960
50 percentile (Median)	12,260
90 percentile	90,444
Standard Deviation	79,300

Table 4: Estimation of the caching performance based on stack depth analysis.

Cache size (TB)	Average # of files in cache	% of requests not fitting
50	132,830	6.15
25	66,415	13.84
10	26,566	31.83
5	13,283	48.40
1	2,656	76.80

all cache sizes. This is an upper bound for filecule prefetching because the filecule used are optimal, obtained by analyzing the entire workload. In Section 4.2 we present the low impact on filecule accuracy (and thus on resource management performance) when using limited history information.

For cache sizes up to 10 TB the benefits of prefetching are evident, since File LRU has lower byte hit rate than GRV and LRU-Bundle. However, for the larger cache sizes of 25 TB and 50 TB (accounting for 6% and respectively 13% of the workload size), the simple LRU cache replacement algorithm with no prefetching and no job reordering outperforms GRV. Because of the overhead introduced by GRV, a slight performance advantage from a simple algorithms as LRU deserves an explanation.

It turns out that the explanation is the old truth of time locality. A stack depth analysis [8, 2] on the entire set of DZero traces shows that all stack depths are smaller than 1 million (Figure 8), which is less than 10% of the total number of file accesses. Strong temporal locality is represented by small stack depth (defined as the number of memory accesses between consecutive accesses to the same memory location). In Figure 8, the thick dark band close to the X axis indicates a large number of small stack depths. Table 3 presents the relevant statistics.

The unexpected better performance of LRU compared to GRV is due to better representation of time locality in LRU. For 25 TB and 50 TB caches, only 6.15% and 13.84% of the stack depths are higher than the average number of jobs whose average data requests can be stored in the cache (see Table 4).

Table 4 shows the number of files that can be accommodated in each cache size and the percentage of stack depth accesses that are greater than the number of files that can be accommodated. The DZero workload contain 996,227 files whose sizes add up to approximately 375 TB. We obtained the fraction of the number of bytes that can be accommodated in each cache size (cache size/375 TB), and used this fraction to calculate the number of files that can be accommodated in the cache. The number of files that can be accommodated for all cache sizes, with the exception of 1 TB, is greater than the median stack depth accessed.

On average, 100,201 distinct files are accessed per month in the traces studied. Figure 8 shows that most of the stack depths are less than this average and the 90th percentile is 90,444. This indicates that most of the files are accessed again within a month.

Figure 9 shows the average percentage of cache change for the cache sizes considered. As now expected, the average percentage of cache change is the lowest, and thus best, for the LRU-Bundle algorithm for all cache sizes. GRV has large cache changes for all

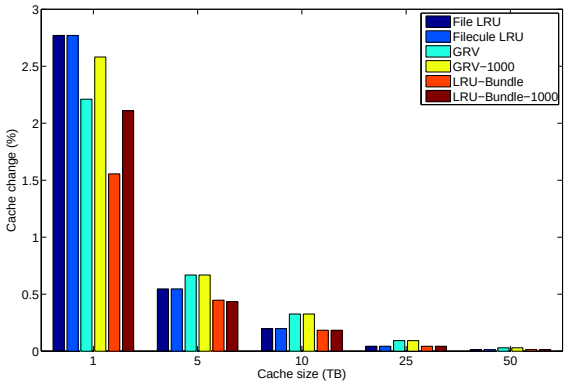

Figure 9: Average percentage of cache change for different cache sizes

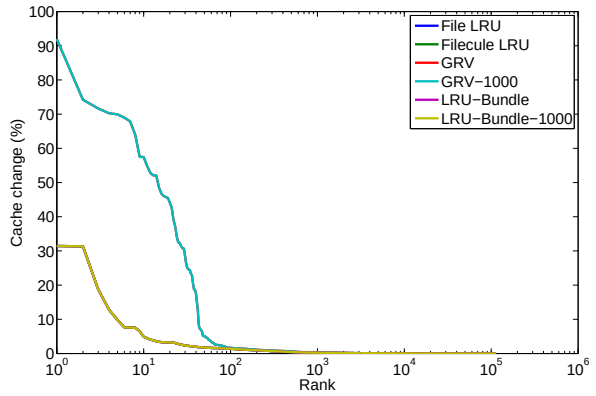

Figure 10: Percentage of Cache Change for Cache Size of 50 TB. File LRU = Filecule LRU = LRU-Bundle = LRU-Bundle-1000 and GRV = GRV-1000

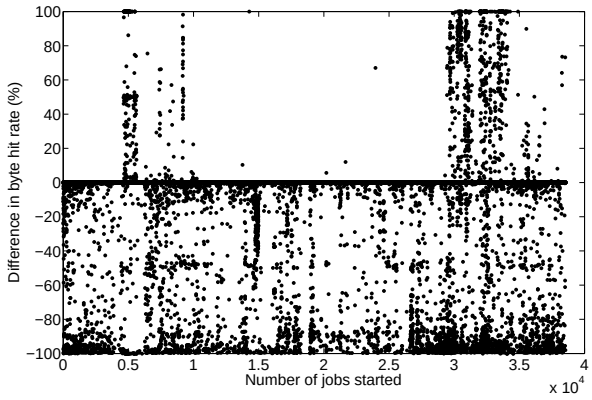

Figure 11: Difference in byte hit rate between filecule LRU with optimal filecules and filecule LRU with 1-month history window. Filecule LRU with optimal filecules has higher byte hit rates for 5,357 jobs (14.9% of the jobs considered). Filecule LRU with 1-month window has higher byte hit rates for 985 jobs (2.5%). Equal byte hit rates observed for 83.5% of the jobs (32,223 jobs).

caches sizes except at 1 TB, where File LRU and Filecule LRU have higher cache changes.

4.2 Lesson 2: Filecule Grouping and the Impact of History Window on Filecule Identification

Filecules are the result of an emergent pattern influenced by data content and user interests. In the current definition, in which filecules are disjoint, identifying them is easy to implement and has relatively low overhead. However, a more relaxed definition of filecules that would allow overlap might turn out more flexible and adaptive (and is in our plans for the future).

Filecule prefetching, as shown in Figure 7 exhibits high byte hit rate compared with the more elaborate GRV algorithm. However, in these experiments we considered optimal filecules, that is, identified from the entire set of traces. This evaluation is optimistic, as it assumes knowledge of the future. How does shorter trace history affect the accuracy of identifying filecules and what are the penalties for staging approximate filecules?

We note that the longer the history available for identifying the filecules, the smaller the filecules resulted. This is because, since filecules are defined as groups of files that are always requested together, new file requests can only break such a grouping by requesting only a subset of its files. A too short request history can thus suggest file groupings larger than needed: larger amounts of data will be prefetched and only part of it will be of use to future requests. The two performance metrics that are most relevant for this study are byte hit rate (as a measure of benefits) and percentage of cache change (as a measure of penalties for predicting a potentially too large set of related files).

We thus evaluate the following:

- since File LRU does not waste any unneeded communication, we compare the percentage of cache change when using filecules identified from the most recent 1-month history and when using file LRU (thus, no prefetching).
- we compare the byte hit rate and percentage of cache change between LRU with optimal filecules and LRU with 1-month history filecules.
- we compare the effect of history length on the accuracy of filecule identification by evaluating the performance of LRU with filecules identified based on 6 months of history.

As expected, our experiments show that the byte hit rate of Filecule LRU using 1-month window is higher than that of File LRU (thus, no prefetching) and lower than that of Filecule LRU using optimal filecules (Figure 11). The former observation is simply due to the benefits of prefetching. The latter is explained by the fact that a file that has never been requested in the 1-month history might be recognized as part of a filecule based on the entire workload, and thus initiate a staging operation in the case of optimal filecule LRU. Unexpectedly, there are cases where the 1-month filecules have better hit rates than the optimal filecules. We explain this behavior by temporal locality in file relationships that suggest shorter history windows may better preserve information.

The percentage of cache change per job when using File LRU and Filecule LRU with optimal filecules is the same (as nothing other than what is needed is ever transferred to the cache). Figure 12 shows the difference between the percentage of cache change between Filecule LRU using 1-month window and optimal Filecule LRU. The difference in percentage of cache change is not substantial: 86% of the jobs have equal cache change, and 8% have an increased cache change for short history window.

As mentioned before, smaller history windows lead to the partitioning of files into fewer but larger filecules. To quantify the impact of the history window length, we compared the effects of 1-month and 6-month windows. Figure 13 shows the difference between byte hit rates obtained using Filecule LRU with 6-month history window and Filecule LRU with 1-month history window. Most of the jobs (92%) have the same byte hit rates for both windows. Interestingly, the filecules identified with 1-month window during November 2003 predict data usage better than the filecules identified with 6-month window. This is perhaps due to an increased activity (50% more jobs were submitted during that month than the monthly average). The difference in percentage of cache change per job between Filecule LRU with 6-month window and 1-month window is below 2.6 (Figure 14).

To summarize, 1-month history is sufficient for determining filecules

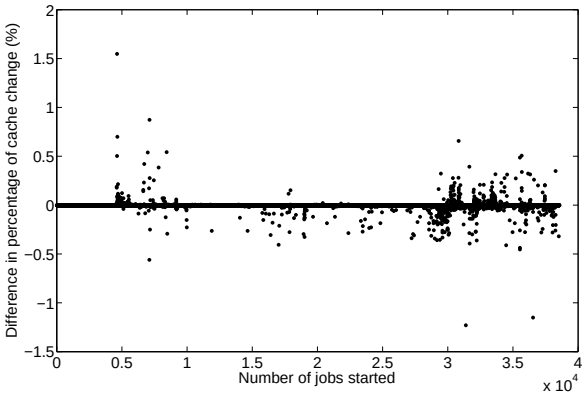

Figure 12: Difference in percentage of cache change between filecule LRU with 1-month Window and Filecule LRU with optimal filecules.

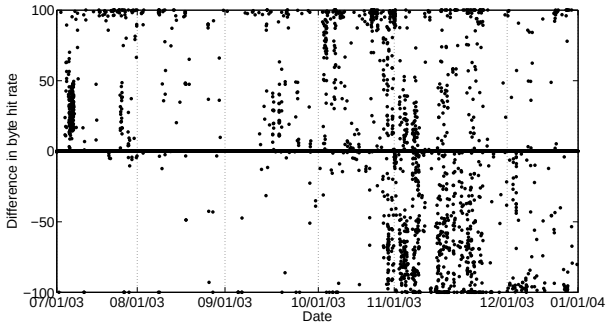

Figure 13: Difference in byte hit rate between filecule LRU with 6-month window and 1-month window. 22,499 (92%) jobs have equal byte hit rates. Filecule LRU with 6-month window has lower byte hit rate for 1,005 jobs (4%). Filecule LRU with 1-month window higher byte hit rate for 888 jobs (3.9%).

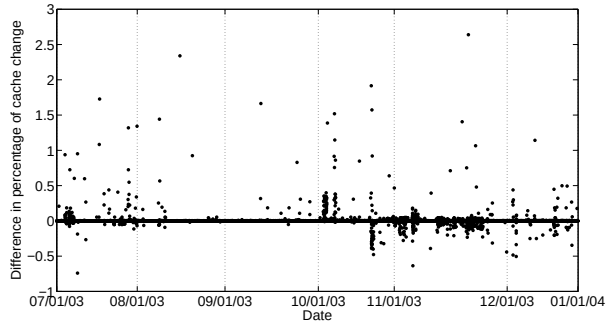

Figure 14: Difference in Percentage of Cache Change Between Filecule LRU with 6-month Window and 1-month Window

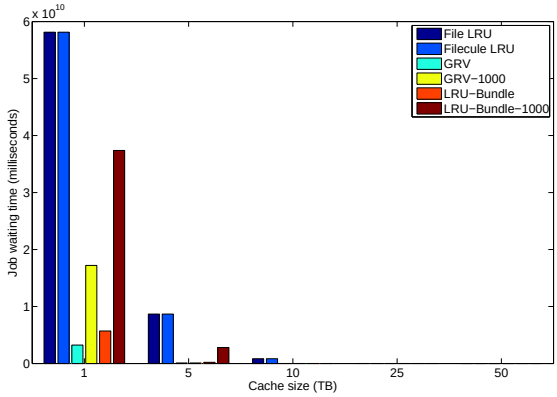

Figure 15: Average Job Waiting Time for Different Cache Sizes

with good accuracy. Given the time locality and the evolution of file relationships, a sliding window is likely to lead to more adaptive grouping of files into filecules. Better than fixing the history window size to a time interval, a window size dictated by the job-inter-arrival time or the transition rate of file popularity [20] may lead to even better results.

4.3 Lesson 3: The Importance of Job Reordering

Our experimental evaluations confirm that data-aware job reordering has high performance impact in data-intensive large-area collaborations. Data-aware job scheduling has been an active topic of research in Grid computing. Our contribution in this respect is two-fold:

1. By using real-world traces from a representative scientific collaboration, our experimental results provide a valid data point in the space of experimental work. The DZero traces dictate parameters that influence job scheduling performance such as job duration times, data requirements, job inter-arrival characteristics, and time locality.

2. Evaluations of job reordering with filecule prefetching. We investigate the filecule prefetching technique coupled with the job scheduling component from GRV and compare its performance with that of a FCFS order. For this, we took the trouble to implement previously proposed solutions to better understand them and accurately compare them with other approaches.

When using FCFS scheduling algorithm, jobs can be delayed only due to lack of available space in the cache to load their data files (we assume no contention on computational resources). For this reason, the job waiting time when FCFS is used coupled with LRU cache replacement policy is independent of data prefetching (thus, File LRU and Filecule LRU will experience identical job delays). In addition to delays due to space constraints, in GRV jobs with lower request values have lower priority and thus are delayed longer.

Figure 15 shows the average job waiting time for the various algorithms for increasing cache sizes. File LRU and Filecule LRU have the worse average waiting times due to the FCFS scheduling order. As expected, queue freezing (marked as LRU-Bundle-1000 and GRV-1000 in the figure) increases the average job waiting time,

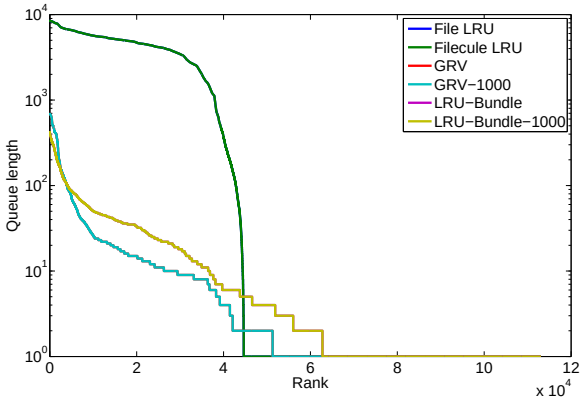

Figure 16: Queue length for cache size of 10 TB. File LRU = Filecule LRU, GRV = GRV-1000 and LRU-Bundle = LRU-Bundle-1000

since new jobs are delayed until all the jobs in the frozen queue are scheduled.

A special discussion is necessary for the case of filecule-based LRU with GRV job reordering, since filecules are dynamically identified based on the previous 1-month history. In order to quantify reduction in data transfer and increase in byte hit rate when using filecules for prefetching for LRU-GRV algorithm, we experimented by simulating filecule identification at the end of each month. We used filecules identified using jobs executed during one month for prefetching data into the cache during the next month. We did this experiment for all the jobs executed during the year 2003 and compare them with the results obtained using the GRV algorithm.

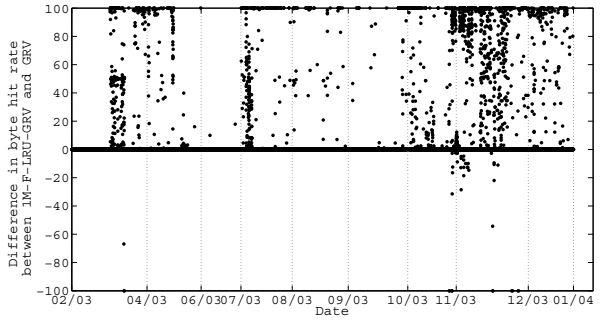

Figure 17: Difference in byte hit rate of LRU-GRV with filecules and GRV

Figure 17 shows the difference over the one-year interval considered in byte hit rate between LRU-GRV with filecules and GRV. It can be observed that there are more data points in the upper section of the plot compared to the lower section. This means that for more jobs the LRU-GRV with filecules provides higher byte hit rates than GRV.

Figure 18 shows the difference in percentage of cache change between LRU-GRV with filecules and GRV. The same behavior is noticed: cache change for LRU-GRV with filecules is less than that of GRV, which indicates less data transfer. On average, for a cache size of 25 TB, the LRU-GRV with filecules algorithm provides about 6% increase in byte hit rate over GRV but with significantly

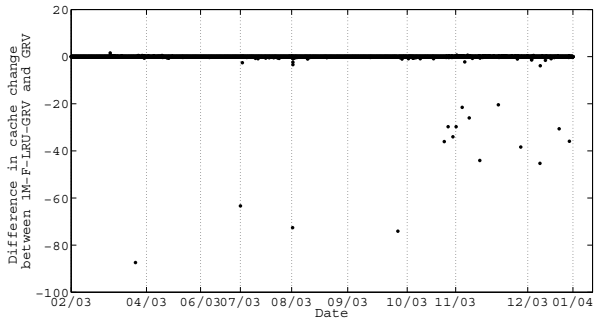

Figure 18: Difference in percentage of cache change of LRU-GRV with filecules and GRV

lower costs (50%, or 12.5 TB of data) in terms of data transfer. This is a significant result as it proves on real traces the advantage of the old, unglamorous LRU algorithm applied to a rather intuitive file grouping approach based on filecule identification.

5. RELATED WORK

Data replication techniques have been studied independently of usage patterns in [31, 45]. Logistical Networks [11, 10] deal with resource management across storage, computation, and data transmission. While relevant to the work discussed here, logistical networks take a more invasive approach by modifying (instead of building on) the existing infrastructure.

Data management in grids is one of the main foci for researchers and users of grid computing: scheduling data-intensive jobs [42, 43, 44, 32] and scheduling data transfers [3, 13] have received significant attention. iRODS [41] is a rule-based data management system currently under investigation that allows data administrators to specify management rules related to the preservation and preprocessing of their data.

Workload characterization in scientific communities have not yet been the focus of in-depth analysis and modeling because only a few wide-area scientific collaborations and only recently have ramped up production runs at significant scales. Results have been published on resource characterization in terms of load [29] and availability [30]. Synthetic workloads have been proposed in [28].

The need for a better understanding of workloads and a set of realistic benchmarks has been presented in [22] in the context of correctly evaluating solutions for decentralized systems.

Workload-aware resource management has been traditionally addressed at the system level in disk management [48, 49, 50, 15]. File grouping techniques have been investigated in the context of mobile computing [33], caching in distributed file systems [6], web proxies [46], disk access [18], and Microsoft Windows updates [21]. Solutions that implement application-aware data management include LOCKSS [35] for library archives and the Google File System [19] for Google data processing.

6. SUMMARY: LESSONS FROM USING REAL-WORLD TRACES

This study presents a set of experimental results using real traces from a large high-energy physics collaboration, the DZero Experiment. Our analysis based on real workloads help us formulate recommendations that are likely to have impact beyond the particular set of traces used or the science domain where they were produced. These recommendations are formulated for (1) modeling

workloads for data-intensive scientific collaborations; and (2) for designing data-management techniques adapted to multi-file processing.

6.1 Workload Modeling

Synthetic workloads are often generated for supporting experimental efforts when appropriate or sufficient real-world workloads are not available. We caution the community about two patterns that have been identified in many systems but that are not found in the DZero traces. We believe it is likely that we notice a behavior that may appear in other application domains as well.

The first such pattern is the file size distribution. We noticed multiple peaks, due to different types of data and the processes that create them. This is likely to be the case in many other domains.

The second workload characteristic that contradicts traditional models is the file popularity distribution. Our analysis shows that files are more uniformly popular than in the Zipf distribution typically considered, which decreases the benefits of caching.

Another observation possibly useful to workload modeling is the strong time locality of the traces analyzed. This observation is strongly related to the design and evaluation of data management techniques.

6.2 Designing Data Management Algorithms for Science Grids

Our experiments with real workloads on multi-file staging and job reordering techniques confirm two common believes. First, preserving time locality in grouping files has significant impact on caching performance. Second, it demonstrates the significant impact of combining job reordering with data placement in data-intensive resource-sharing environments.

In addition, we proposed and evaluated a new combination of file staging with job scheduling referred to as LRU-Bundle (Section 4). We compared the performance of LRU-Bundle with LRU and GRV cache replacement algorithms by simulating disk cache events using real traces from the DZero Experiment. Our experiments show that LRU-Bundle provides better byte hit rates compared to File LRU (4%-106%) and GRV (4%-8%), but most importantly, it requires significantly less data transfer (30% to 56%) compared to GRV. When filecules are used for prefetching, LRU-Bundle algorithm further improves the byte hit rate and reduces the volume of data moved.

Filecules can be also applied to data replication, job scheduling, resource selection, and data staging. By using filecules for data replication, related data can be stored together at the same location. This ensures faster data search and retrieval. Moreover, it can guide job scheduling for selecting a computational resource close to where the data needed by the job is stored.

The degree of correlation between filecules can be utilized for data staging. A threshold of correlation can be used to determine how far away two filecules should be stored. If the threshold of correlation is met, the filecules are stored in nearby locations. Future work includes such explorations.

7. REFERENCES

[1] The DZero Experiment, http://www-d0.fnal.gov.

[2] S. Acharya, B. Smith, and P. Parnes. Characterizing user access to videos on the world wide web. In *Proceedings of Multimedia Computing and Networking*, 2000.

[3] M. Allen and R. Wolski. The Livny and Plank-Beck problems: Studies in data movement on the computational grid. In *Supercomputing*, 2003.

[4] V. Almeida, A. Bestavros, M. Crovella, and A. deOliveira. Characterizing reference locality in the www. Technical report, Boston, MA, USA, 1996.

[5] A. Amer, D. D. E. Long, and R. C. Burns. Group-based management of distributed file caches. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 525, Washington, DC, USA, 2002. IEEE Computer Society.

[6] A. Amer, D. D. E. Long, and R. C. Burns. Group-based management of distributed file caches. In *ICDCS*, 2002.

[7] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a web proxy in a cable modem environment. *SIGMETRICS Perform. Eval. Rev.*, 27(2):25–36, 1999.

[8] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. Technical report, 1999.

[9] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns: Characteristics and caching implications. Technical report, Boston, MA, USA, 1998.

[10] A. Bassi, M. Beck, J.-P. Gelas, L. Lefèvre, T. Moore, J. Plank, and P. V.-B. Primet. Active and logistical networking for grid computing: the e-toile architecture. *Future Gener. Comput. Syst.*, 21(1):199–208, 2005.

[11] M. Beck, T. Moore, and J. S. Plank. An end-to-end approach to globally scalable network storage. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 339–346, New York, NY, USA, 2002. ACM Press.

[12] A. Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *SPDP '95: Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, page 338, Washington, DC, USA, 1995. IEEE Computer Society.

[13] U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. Scheduling file transfers for data-intensive jobs on heterogeneous clusters. In *Proceedings of the 13th European Conference on Parallel and Distributed Computing (Europar)*, 2007.

[14] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of www client-based traces. Technical report, Boston, MA, USA, 1995.

[15] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. Diskseen: Exploiting disk layout and access history to enhance I/O prefetch. In *USENIX Annual Technical Conference*, 2007.

[16] S. Doraimani. Filecules: A new granularity for resource management in grids. Master's thesis, University of South Florida, 2007.

[17] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *SIGMETRICS '99: Proceedings of the 1999 ACM SIGMETRICS International conference on Measurement and Modeling of Computer Systems*, pages 59–70, New York, NY, USA, 1999. ACM Press.

[18] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files. In *USENIX Annual Technical Conference*, pages 1–17, 1997.

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM Press.

[20] A. S. Gish, Y. Shavitt, and T. Tankel. Geographical statistics and characteristics of p2p query strings. In *IPTPS2007 - Proceedings of the 6th International Workshop on Peer-to-Peer Systems*, February 2007.

[21] C. Gkantsidis, T. Karagiannis, and M. Vojnovic. Planet scale software updates. In *SIGCOMM*, pages 423–434, 2006.

[22] A. Haeberlen, A. Mislove, A. Post, and P. Druschel. Fallacies in evaluating decentralized systems. In *The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[23] A. Iamnitchi, S. Doraimani, and G. Garzoglio. Filecules in high-energy physics: Characteristics and impact on resource management. In *HPDC-15: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, pages 69–79, June 2006.

[24] A. Iamnitchi and I. Foster. Interest-aware information dissemination in small-world communities. In *High Performance Distributed Computing*, 2005.

[25] A. Iamnitchi and M. Ripeanu. Myth and reality: Usage behavior in a large data-intensive physics project, 2003.

[26] A. Iamnitchi, M. Ripeanu, and I. Foster. Small-world file-sharing communities. In *Infocom*, Hong Kong, China, 2004.

[27] A. Iosup, D.H.J.Epema, P. Couvares, A. Karp, and M. Livny. Build-and-test workloads for grid middleware: Problem, analysis, and applications, 2007.

[28] A. Iosup and D. Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 313–320, Washington, DC, USA, 2006. IEEE Computer Society.

[29] A. Iosup, M. Jan, O. Sonmez, and D. Epema. The characteristics and performance of groups of jobs in grids. In *EuroPar*, 2007.

[30] A. Iosup, M. Jan, O. Sonmez, and D. Epema. On the dynamic resource availability in grids. In *Grid,*, 2007.

[31] M. Karlsson and C. Karamanolis. Choosing replica placement heuristics for wide-area systems. In *The 24th International Conference on Distributed Computing Systems (ICDCS)*, March 2004.

[32] G. Khanna, N. Vydyanathan, U. Catalyurek, T. Kurc, S. Krishnamoorthy, P. Sadayappan, and J. Saltz. Task scheduling and file replication for data-intensive jobs with batch-shared I/O. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2006.

[33] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 264–275, New York, NY, USA, 1997. ACM Press.

[34] L. Loebel-Carpenter, L. Lueking, C. Moore, R. Pordes, J. Trumbo, S. Veseli, I. Terekhov, M. Vranicar, S. White, and V. White. Sam and the particle physics data grid. In *In Computing in High-Energy and Nuclear Physics, Beijing, China*, 2001.

[35] P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. H. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. *ACM Trans. Comput. Syst.*, 23(1):2–50, 2005.

[36] E. Otoo, D. Rotem, and A. Romosan. Optimal file-bundle caching algorithms for data-grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 6, Washington, DC, USA, 2004. IEEE Computer Society.

[37] E. Otoo, D. Rotem, and A. Romosan. Optimal file-bundle caching algorithms for data-grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 6, Washington, DC, USA, 2004. IEEE Computer Society.

[38] E. Otoo, D. Rotem, A. Romosan, and S. Seshadri. File caching in data intensive scientific applications. In *Data Management in Grids - Lecture Notes in Computer Science*, volume Volume 3836/2006, pages 85–99. Springer Berlin / Heidelberg, 2006.

[39] E. Otoo, D. Rotem, and S. Seshadri. Efficient algorithms for multi-file caching. In *Database and Expert Systems Applications - Lecture Notes in Computer Science*, pages 707–719. Springer Berlin/Heidelberg, 2004.

[40] A. Rajasekar, M. Wan, R. Moore, G. Kremenek, and T. Guptil. Data grids, collections, and grid bricks. In *MSS '03: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, page 2, Washington, DC, USA, 2003. IEEE Computer Society.

[41] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder. A prototype rule-based distributed data management system. In *Workshop on Next Generation Distributed Data Management*, 2006.

[42] K. Ranganathan and I. Foster. Design and evaluation of dynamic replication strategies for a high performance data Grid. In *International Conference on Computing in High Energy and Nuclear Physics*, 2001.

[43] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for a high performance data Grid. In *International Workshop on Grid Computing*, 2001.

[44] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, 2002.

[45] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *OSDI*, 2002.

[46] E. A. M. Shriver, E. Gabber, L. Huang, and C. A. Stein. Storage management for web proxies. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 203–216, Berkeley, CA, USA, 2001. USENIX Association.

[47] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability, 2001.

[48] C. Staelin and H. Garcia-Molina. Clustering active disk data to improve disk performance. Technical Report CS–TR–298–90, Princeton, NJ, USA, 1990.

[49] C. Staelin and H. Garcia-Molina. Smart filesystems. In *USENIX Winter*, pages 45–52, 1991.

[50] C. D. Tait and D. Duchamp. Detection and exploitation of file working sets. In *Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS)*, pages 2–9, Washington, DC, 1991. IEEE Computer Society.

[51] I. Terekhov. Meta-computing at d0. In *In Nuclear Instruments and Methods in Physics Research, Section A, NIMA14225*, volume 502/2-3, pages 402–406, 2002.