Cluster Comput (2009) 12: 153–173
DOI 10.1007/s10586-009-0081-3

Workload characterization in a high-energy data grid and impact on resource management

Adriana Iamnitchi · Shyamala Doraimani · Gabriele Garzoglio

Received: 1 January 2009 / Accepted: 6 January 2009 / Published online: 23 January 2009

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.

Keywords Grid computing · Workload characterization · Data management

A. Iamnitchi (✉) · S. Doraimani
Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA
e-mail: anda@cse.usf.edu

S. Doraimani
e-mail: sdoraimani@gmail.com

G. Garzoglio
Computing Division, Fermi National Accelerator Laboratory, Batavia, IL 60510, USA
e-mail: garzoglio@fnal.gov

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.

While it is acknowledged that scientific applications often process multiple input files [2, 39], few studies [15, 16, 27] provide a quantitative characterization of this pattern in scientific workloads. In addition, efforts to provide benchmarks for Grid workloads [29–31] have been challenged by 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 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 [18], hosted at the Fermi National Accelerator Laboratory.

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 new data abstraction, *filecule* [fil'-eh-kyul'], defined as a group of files always accessed together. Our analysis of the DZero traces shows the existence of filecules in this workload.
– An experimental evaluation of the impact of filecules for data management. 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.
– A set of design recommendations for data staging for distributed collaborations, such as preserving time locality and enforcing the role of job reordering.
– 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 presents the overall characteristics of the DZero workloads. Section 3 presents our analysis of this workload from the perspective of data usage. It introduces filecules and evaluates the size, popularity and lifetime distributions for both files and filecules. Section 4 describes the simulator we built and the set of data management algorithms implemented. Experimental results are presented in Sect. 5 as a set of lessons inferred from our study. Section 6 reviews related work on file grouping and techniques for data management that consider file groupings. Finally, Sect. 7 summarizes our results and describes future work.

2 The DZero workload

In Grid terminology [19], the DZero Experiment [18] is a virtual organization that provides a worldwide system of shareable computing and storage resources to hundreds of physicists in more than 70 institutions from 18 countries. DZero is one of the two experiments currently processing data from the Tevatron collider at the Fermi National Accelerator Laboratory. The DZero community studies particles formed from the annihilation of protons and antiprotons at the TeV energy scale. The signals recorded at different layers of the detector form a physics event. Events consist of about 250 KB of information and are stored in "raw" data files of about 1 GB in size. Every bit of raw data is accessed for further processing and filtering, then the resulting data is classified based on the physics events they represent.

DZero is representative for data grids in general because it is a globally-distributed scientific community who shares a mature software and hardware infrastructure that has been running in production mode for many years. This infrastructure supports a variety of scientific applications specific to high-energy physics, with different resource and service needs. This study provides a representative data point in the space of deployed grid infrastructures that serve science communities. However, we refrain from generalizing the results of our study, since we are aware of the huge variety of data-intensive applications supported by grid computing technologies. Yet, we discover workload characteristics that contradict well-accepted patterns in distributed systems. In this case, a thorough analysis of one community provides sufficient evidence to question the validity of accepted general patterns and thus better inform workload models and consequently the design and evaluation of related techniques.

2.1 The DZero experiment

Modern high-energy physics experiments, such as DZero, typically acquire more than 1 TB of data per day and move up to ten times as much. In addition to the stream of data from the detector, various other computing activities contribute to the 1 TB of derived and simulated data stored per day. Three main activities take place within the DZero Experiment: data filtering (called *data reconstruction* in the DZero terminology), the production of simulated events, and data analysis. The first two activities are indispensable for the third one. During data reconstruction, the binary format of every event from the detector is transformed into a format that more easily maps to abstract physics concepts, such as particle tracks, charge, or spin. The production of simulated events, also called Monte Carlo production, is necessary for understanding and isolating the detector characteristics related to hardware, such as the particle detection efficiency, or to physics phenomena, such as signal to background discrimination. Data analysis mainly consists of the selection and the statistical study of particles with certain characteristics.

Table 1 Characteristics of workloads per data tier

Data tier	Users	Jobs	Files	# of file requests	Data input/job (GB)		Job run time (hours)	
					Avg.	Std. dev.	Avg.	Std. dev.
Reconstructed	304	17,552	507,796	1,770,176	34	285	11.08	38.52
Root-tuple	51	1,226	59,923	468,176	85	115	14.19	28.93
Thumbnail	440	94,284	428,508	9,329,734	50	319	8.08	28.83

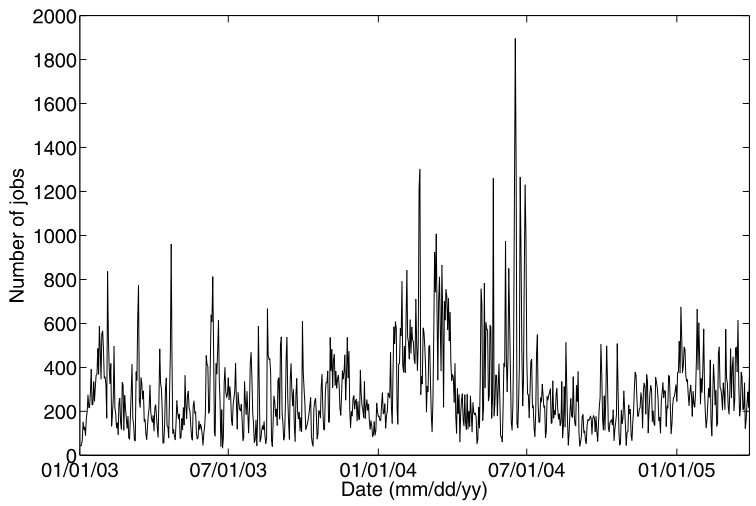


Fig. 1 Number of jobs submitted each day

Tracing system utilization is possible via a software layer named SAM [35, 55] that provides centralized file-based data management. The SAM system offers four main services: first, it provides reliable data storage, either directly from the detector or from data processing facilities around the world. Second, it enables data distribution to and from all of the collaborating institutions. Third, it thoroughly catalogs data for content, provenance, status, location, processing history, user-defined datasets, and so on. And finally, it manages the distributed resources to optimize their usage and enforce the policies of the experiment.

SAM categorizes typical high energy physics computation activities in application families (reconstruction, analysis, etc.). Applications belonging to a family are identified by a name and a version. This categorization is convenient for bookkeeping as well as for resource optimization. Due to the data intensive nature of the high energy physics domain, applications almost always process data. Such data is organized in tiers, defined according to the format of the physics events. Relevant data tiers, some of which are discussed in this work, are the raw, reconstructed, thumbnail, and root-tuple tiers. The raw tier identifies data coming directly from the detector. The reconstructed and thumbnail tiers identify the output of the reconstruction applications in two different formats. The root-tuple tier identifies typically highly processed events in root format [10] and are generally input to analysis applications.

For the data handling middleware, an application running on a *dataset* (set of input files) defines a *job*. Jobs are initiated by a user on behalf of a physics group and typically trigger data movement.

2.2 Workload characteristics

The studies presented in this article are based on SAM logs recorded between January 2003 and March 2005. The traces include information about files and jobs. File information contains file size, file name, and which job requested the file as input. Job information includes description of the application (such as application name, version, and family), its corresponding data tier, user name and group that initiated the job and the start/end time of the job. From the 234,069 jobs submitted during the period of our workloads, we focus on the 113,062 jobs that use data from the reconstructed, root-tuple and thumbnail data tiers, the most data-intensive data tiers in the DZero system (see Table 1). The total number of file requests generated by these jobs is 11,568,086. These requests are for 996,227 distinct files that overall consume 375 TB of storage.

Figure 1 shows the number of jobs submitted each day while Table 2 presents the overall statistical data. Note that DZero jobs typically process multiple input files (Fig. 2): 102 files on average, with a median value of 12.

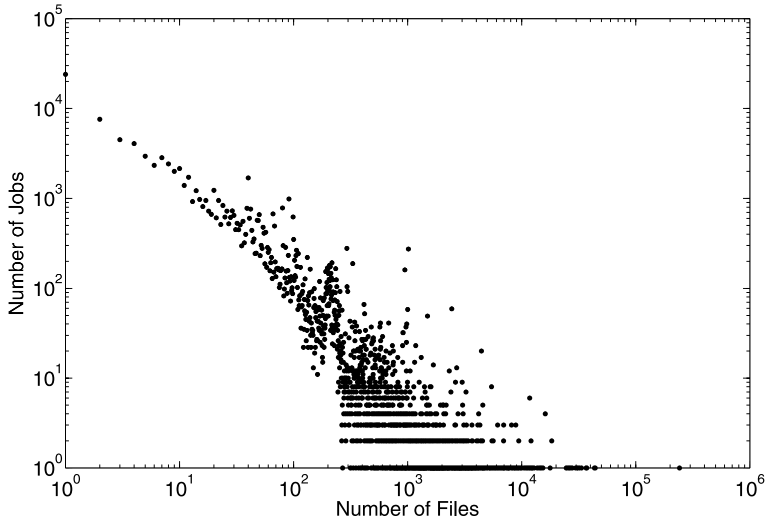


Fig. 2 Number of files requested per job

Table 2 Statistics on data usage

Category	Mean	Median
Jobs per day	137	107
Files per job	102	12
Files accessed per day	15,199	10,853
Distinct files accessed per day	9,534	7,318
Data accessed per day (TB)	6.86	5.08
Storage accessed per day (TB)	4.54	3.45

3 Data usage characteristics

Data is one of the most important resources in Grid collaborations. It is often considered the cause of performance bottlenecks in large-scale collaborations as it consumes a huge set of resources in storage and network bandwidth.

This section quantifies the most important characteristics of data usage in DZero. While these characteristics cannot be directly generalized to other application domains, they can be used to direct the generation of synthetic, realistic workloads. In particular, our analyses corrects generalizations from other systems, such as operating systems or the Web, by showing, for example, that the file popularity distribution in DZero is not Zipf and the file size distribution does not follow a log-normal distribution. However, this study also confirms other patterns from systems research, such as time locality in data usage (as seen from the stack-depth analysis presented later in Sect. 5.1).

In addition, in this section we quantify and characterize the properties of file groupings inferred from data usage. While it is accepted that scientific data analysis applications access and process groups of files, the analysis of the DZero traces [15, 27] is the first that quantitatively characterizes these groupings in the context of science Grids. Specifically, we define a way to group files into disjoint sets based on their usage simultaneity and evaluate the characteristics of these groups in terms of size, popularity, lifetime, and the correlation between these attributes.

3.1 Filecules

The pattern of scientific jobs accessing multiple input files, as presented in Fig. 2, has been acknowledged in previous work [38]. However, to the best of our knowledge, no quantitative characterization of this pattern has been performed. In order to quantify the simultaneous use of groups of files for one job, we separate such a group in disjoint sets of files that we coin *filecules* [fil'-eh-kyuls'].

Inspired from the definition of a molecule, we define 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.

Formally, a set of files $f_1, \ldots, f_n$ 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. Thus, popularity distribution on files and filecules is the same.
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 traces.

Table 3 Statistics of size, popularity and lifetime for files and filecules. Overall in the traces analyzed there are 996,227 files and 96,454 filecules. The number of total requests for files is 11,568,086 and for filecules is 3,918,553

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

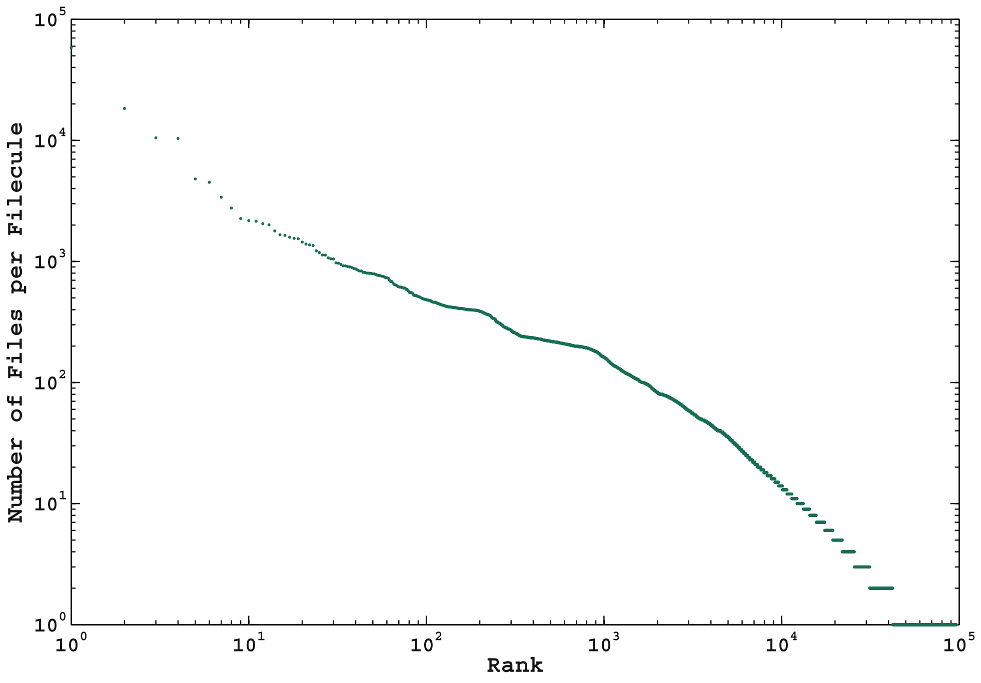


Fig. 3 Number of files per filecule

We stress that filecules can only be identified based on traces of co-occurrence of input files in a job. The entire dataset of the first job in the trace will be initially considered as forming a filecule. Upon the arrival of a new job, its dataset is checked against the current filecule definitions. If non-empty, the intersection between the dataset and each of the filecules will overwrite the filecule definition with possibly two new filecules: the set of files in the intersection and the set of files in the initial filecule that are disjoint from the files in the dataset. Note that this last set can be empty, in which case the initial filecule definition satisfy the property (1) from above and is not broken into smaller sets.

The 27 months of DZero traces revealed 96,454 filecules requested on average more than 40 times each (3,918,553 filecule requests). Table 3 presents the size, popularity and lifetime for files and filecules. Figure 3 shows the distribution of the number of files per filecule: about 56% are one-file filecules. In terms of size, about 5% of the filecules are above 15 TB.

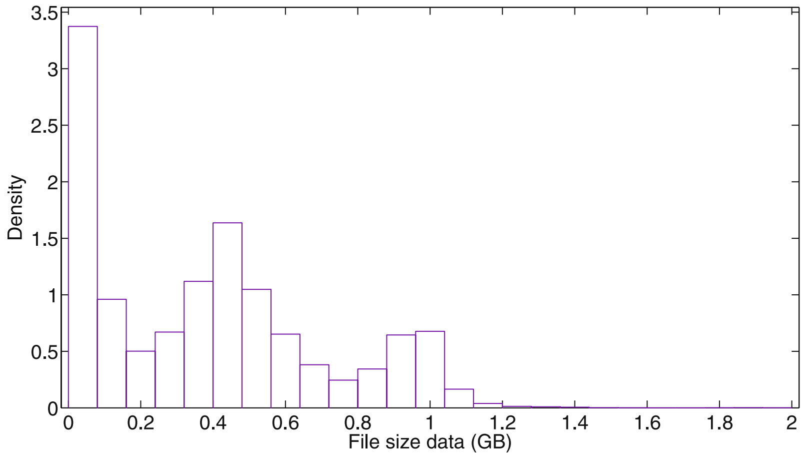
From the perspective of data usage, filecules are a more accurate grouping of data than files: for example, the size of a file can be dictated by filesystem-imposed limitations. Grouping data based on usage can have significant consequences on the performance of caching, prefetching, scheduling data transfers and data replication in distributed systems. Section 5 presents the advantages of using filecules for data prefetching.

3.2 File and filecule size distribution

In data-sharing community, file size characteristics may determine the performance of many resource management techniques, such as caching performance on local storage, or provisioning at storage and networking level. In the absence of real workloads for in-production grids, researchers assume patterns seen in similar previous systems. A natural file size distribution model used is the log-normal distribution, as seen in file systems [17]. In this section we test the validity of this assumption at file and filecule level.

Figure 4 shows the size distribution of files in the DZero workload: the smallest file is 234 bytes and the largest is 2.1 GB. The mean and median files are 0.4 GB. About 69% of files are smaller than 110 MB. The next popular file sizes are around 500 MB and 1 GB.

Fig. 4 File size distribution



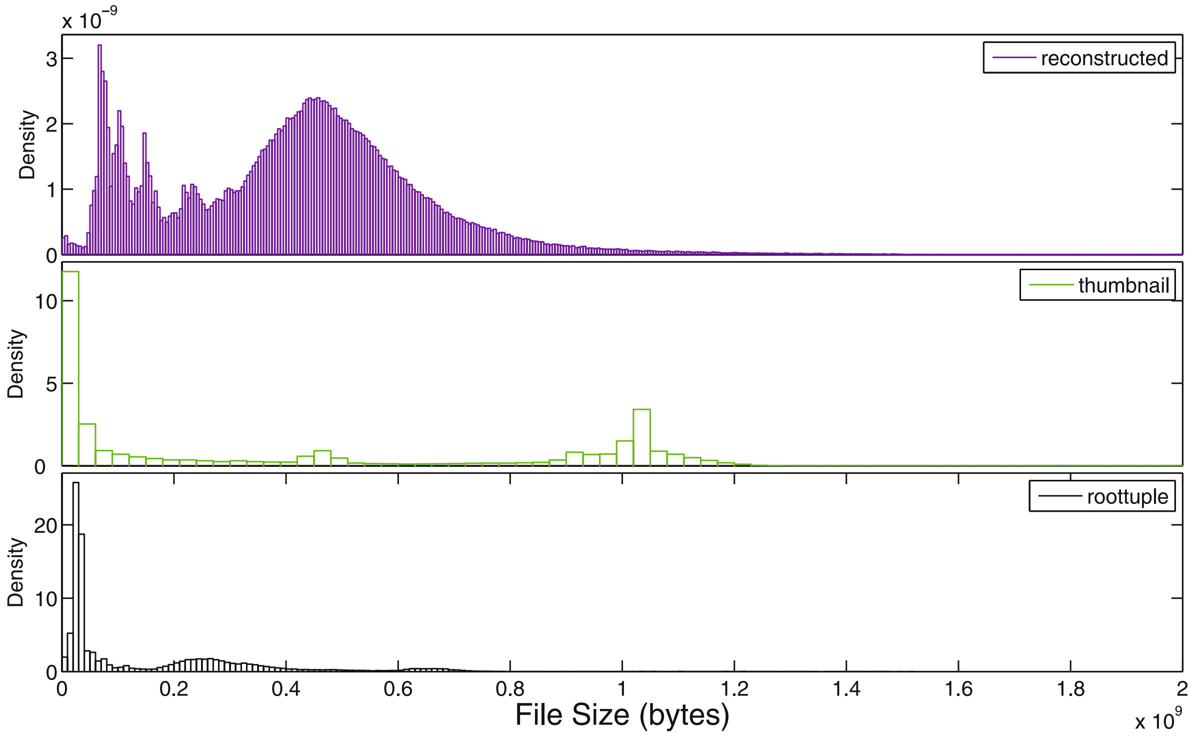


Fig. 5 File size distribution per data tier

The multiple peaks observed in the file size distribution is due to the different peaks observed in different data tiers as shown in Fig. 5. The files in different data tiers are generated as a result of various reconstruction applications, which consequently lead to different file sizes.

Figure 6 shows the filecule size distribution, with sizes ranging from 23.5 KB to 15.7 TB. The files smaller than 23.5 KB (2,264 files) have been grouped into filecules. The largest filecule is about 8,000 times the largest file size and contains 18,326 files. The mean filecule size is 4.2 GB and the median filecule is 1.1 GB. The difference between mean and median is due to a few large filecules (5% have a size larger than 15 TB).

The distribution that best fits the filecule size distribution is the log-logistic distribution with parameters mentioned in Fig. 6. This contradicts the log-normal size distribution of data observed in Windows file systems [17] and web client traces [8]. 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.

3.3 Popularity

The popularity of a file is considered being the number of times the file has been requested. As mentioned before, be-

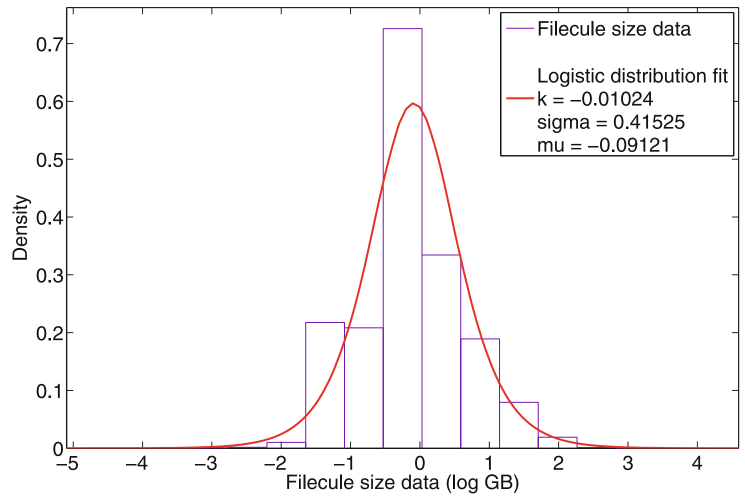


Fig. 6 Filecule size distribution

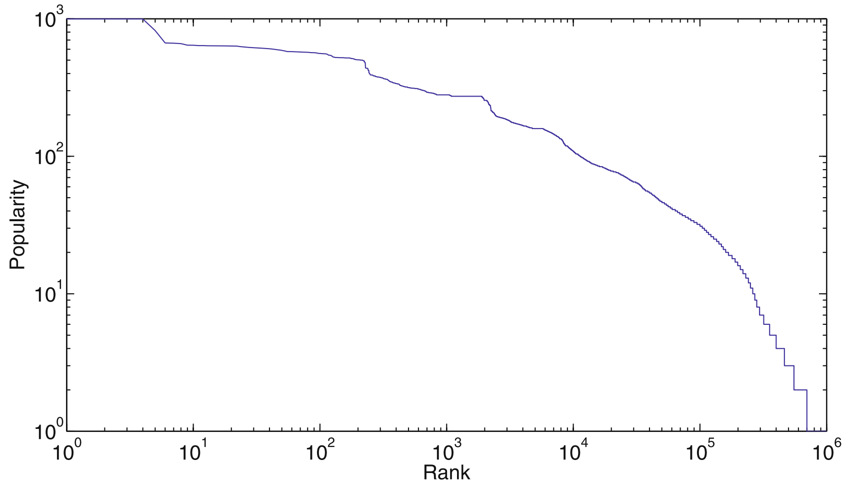


Fig. 7 File popularity vs. rank

cause of how filecules are defined, the number of requests for a file is identical to the number of requests for the filecule that includes that file. However, the popularity distributions of files and filecules will be different since the number of filecules is much smaller than the number of files. Since the popularity distribution is particularly relevant for predicting caching performance, and because we are interested in understanding the effect of using filecules as a data abstraction for data management, we analyze the popularity distributions for both files and filecules.

Figures 7 and 8 present the popularity distributions for files and, respectively, filecules in the traditional log-log plot of the number of requests ordered from most popular to least. Both distributions differ from the traditionally-accepted Zipf distribution observed in [3, 13] and [6]: their tail is much shorter. Moreover, filecule popularity is not as heavy tailed as file popularity. The explanation is that files with less popularity group better into filecules than the highly popular files. This observation is similar to the discussion in [4], which notes 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.

The most popular file was requested by 34 different users in 996 jobs. About 30% of the files have been used by only one job (file popularity = 1) while 6.5% of the total number of files (65,536 files) account for 45% of total requests. For comparison, 6.5% (6,270) of the filecules account for 33.3% of total requests. This is similar to the observation on web traces [7] where a small set of files account for the majority of the requests. Only 4 files (forming one filecule) have the highest popularity of 996.

3.4 Lifetime

We define the lifetime of a file or filecule as the time difference between the start time of the first job that accessed the file and the end time of the last job that accessed it in our trace. Figure 9 presents the file lifetime distribution: 40% (396,341) of the files have lifetime shorter than one week. The median file lifetime is approximately 1 month (712 hours). About 35% of the files (or 70% of filecules)

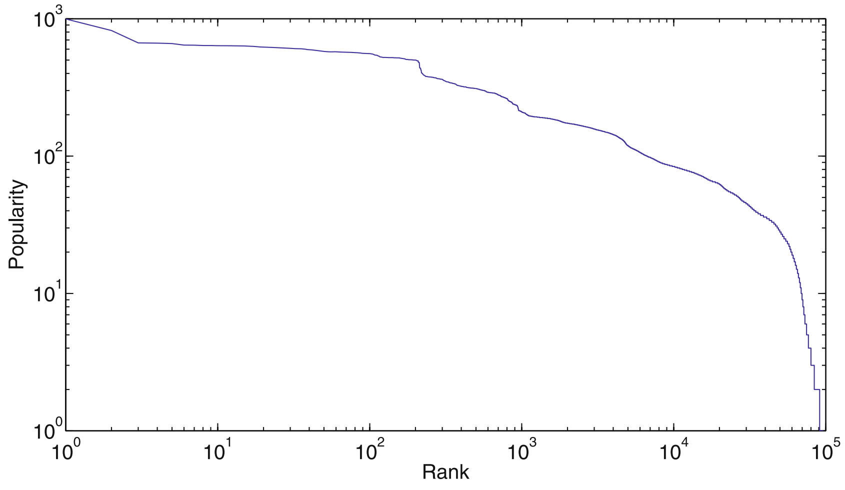


Fig. 8 Filecule popularity vs. rank

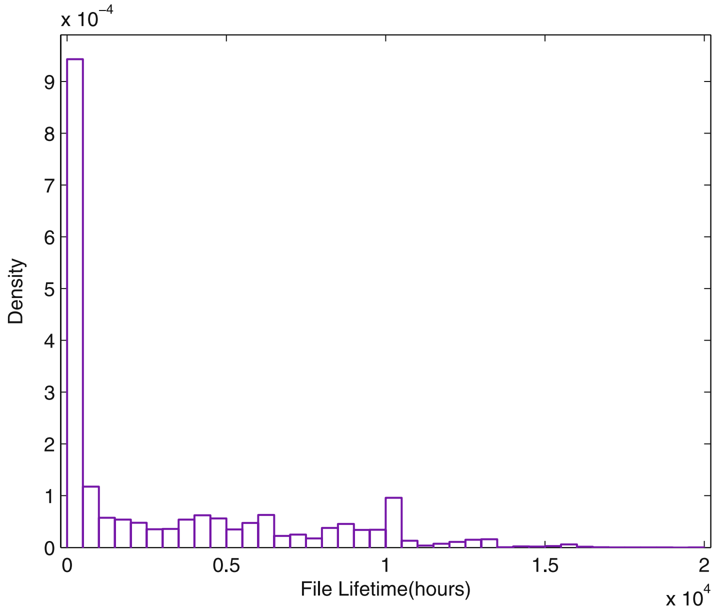


Fig. 9 File lifetime distribution

have a lifetime greater than or equal to 5 months. This behavior is more pronounced than in a web proxy workload [6], where 20% of the files were active after 5 months. 30% of files (equivalent to 5% of filecules) have a lifetime of less than or equal to a day (24 hours). This number is less than the ones mentioned in [6] which reports 50% of inactive files on the next day. This indicates that the files in DZero have longer lifetimes than web pages.

These results impact the effectiveness of caching. On average, 4.54 TB of distinct data are requested each day and 30% of this data becomes useless on the next day, leaving 70% of data still active. In order for a cache to be effective, it needs to retain around 70% of the data from the previous day. This observation implies that any cache size that is less than 3.5 TB (70% of 5 TB) will be too small to take advantage of the temporal locality over more than 24 hours.

Figure 10 presents the filecule lifetime distribution, that follows an extreme value distribution with shape parameter $k = -0.1539$, scale parameter $sigma = 3{,}649.1$ and location parameter $\mu = 4{,}612$. This result differs from the hyperexponential distribution observed for Windows files [17].

The disproportion between filecules corresponding to long-lived files (70% of filecules correspond to 35% of files) and filecules corresponding to short-lived files (5% of filecules correspond to 30% of files) may suggest that files with short active periods have a better tendency to group than files that have long active periods. However, this may be due to the way we identify filecules over long periods of time: a long-lived file is likely to appear in the company of different files over longer intervals and will likely lead to refined, smaller filecules. A question arises from this discussion: what is a good time interval to consider for identifying

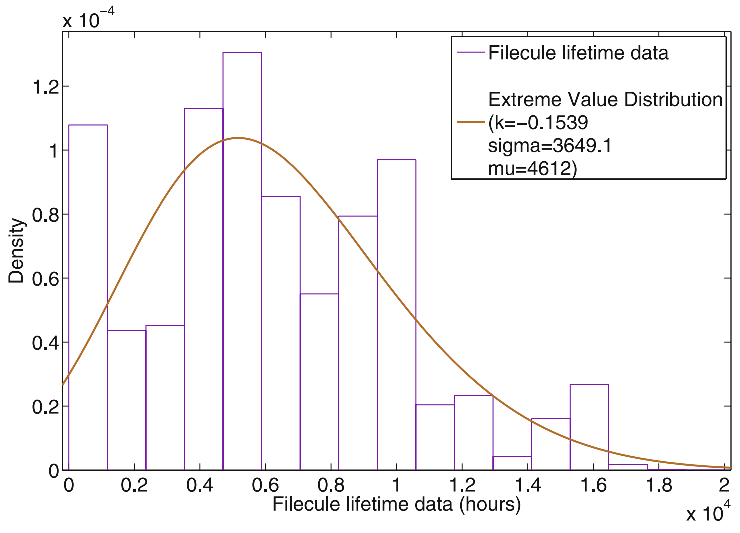


Fig. 10 Filecule lifetime distribution

Table 4 Coefficients of correlation for file and filecule attributes

File size & File popularity	0.1235
File size & File lifetime	−0.0172
File popularity & File lifetime	0.3888
Filecule size & Filecule popularity	−0.0390
Filecule size & Filecule lifetime	−0.0602
Filecule popularity & Filecule lifetime	0.3989

relevant file groups, or in this case, filecules? We will return to this discussion in Sect. 5.2.

Is there any correlation between size and popularity, or size and lifetime, or, more intuitively, lifetime and popularity? Table 4 answers negatively these questions: There is no correlation between these characteristics, which indicates that using data size or lifetime to make decisions about data popularity is incorrect. For example, evicting data from the cache based on size or time elapsed since first access will not improve the performance of the cache.

4 Caching and job reordering: 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 also evaluate the potential of filecules for data prefetching and compare it with other relevant techniques.

To this end, we implemented a set of data staging and job scheduling algorithms presented in Sect. 4.1 in a simulator described in Sect. 4.3. The experiments were performed using the DZero traces and evaluated based on the performance metrics presented in Sect. 4.2.

4.1 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 and 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. [39–41]. In GRV files are staged based on a *request value* that captures historical usage information such as file popularity and normalized size as described later.

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 the queue.

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 5.

The Greedy Request Value (GRV) algorithm combines cache replacement with job scheduling with the purpose of

Table 5 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

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 $n(f_i)$ (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 (2). Jobs are scheduled in a priority scheduling manner with higher priority associated to jobs with higher relative value V_r.

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

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

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 queue. Job queue freezing can be used to limit the computational overhead, the risk of 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 show that the LRU-Bundle combination is highly efficient, as it takes advantage of workload characteristics (specifically, time locality) while making good use of the cache status (via GRV job reordering).

4.2 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.

4.3 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 Sect. 4.1 and compared them using the metrics listed in Sect. 4.2.

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.

5 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.

5.1 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 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 11 presents the byte hit rates of all the algorithms for different cache sizes. Filecule LRU has the highest byte hit rate for all cache sizes. This is an upper bound for filecule prefetching because the filecule used are optimal, obtained by analyzing the entire workload. In Sect. 5.2 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 algorithm such as LRU deserves an explanation.

It turns out that the explanation is the old truth of time locality. A stack depth analysis on the entire set of DZero

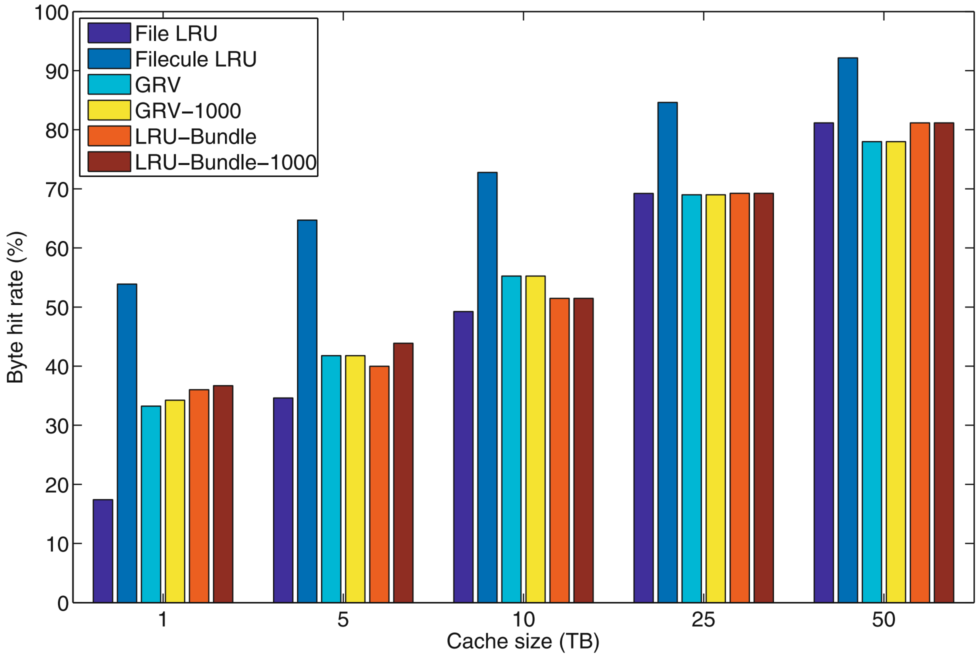


Fig. 11 Average byte hit rate. Cache sizes correspond, in increasing order, to 0.3%, 1.3%, 2.6%, 6% and 13%, respectively, of the total amount of data accessed

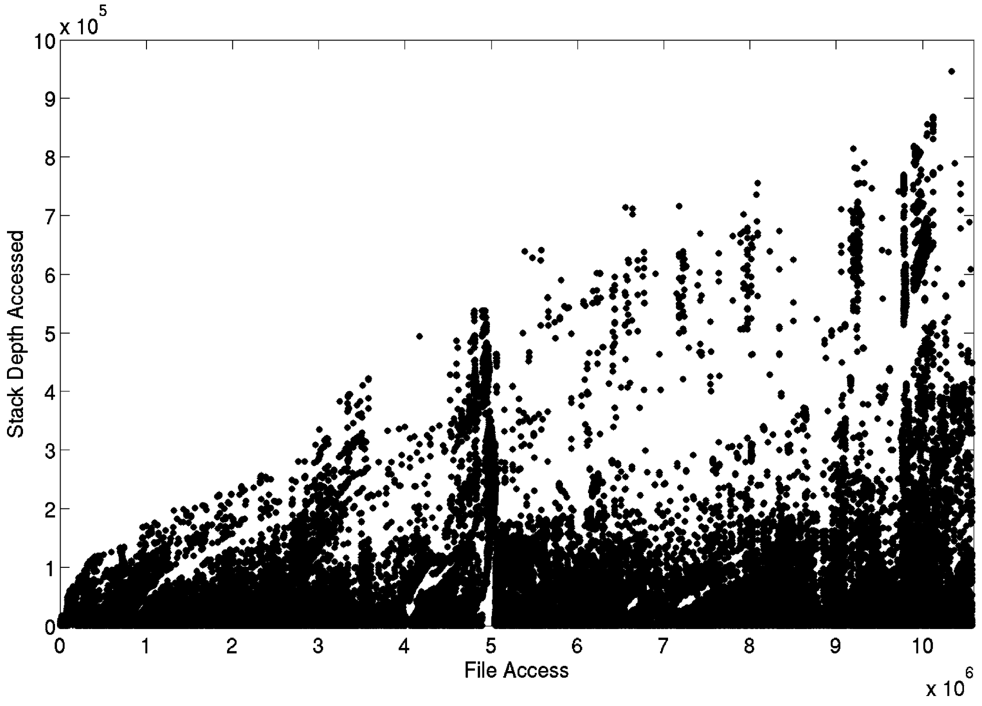


Fig. 12 Stack depth analysis of file requests

Table 6 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 7 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

traces shows that all stack depths are smaller than 1 million (Fig. 12), which is less than 10% of the total number of file accesses. Stack depth is defined as the number of memory accesses between consecutive accesses to the same memory location (note that in our context, a reference to the same memory location is a reference to the same file). Strong temporal locality is represented by small stack depth. The thick dark band close to the X axis indicates a large number of small stack depths. Table 6 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 7).

Table 7 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 12 shows that most of the stack depths are less than this average (the 90th percentile is 90,444). This confirms the lifetime analysis presented above: many files are accessed again within a month.

Figure 13 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 cache sizes except 1 TB, where File LRU and Filecule LRU have higher cache changes.

Figure 14 presents the percentage of cache changes per job in decreasing order for the largest cache size (50 TB).

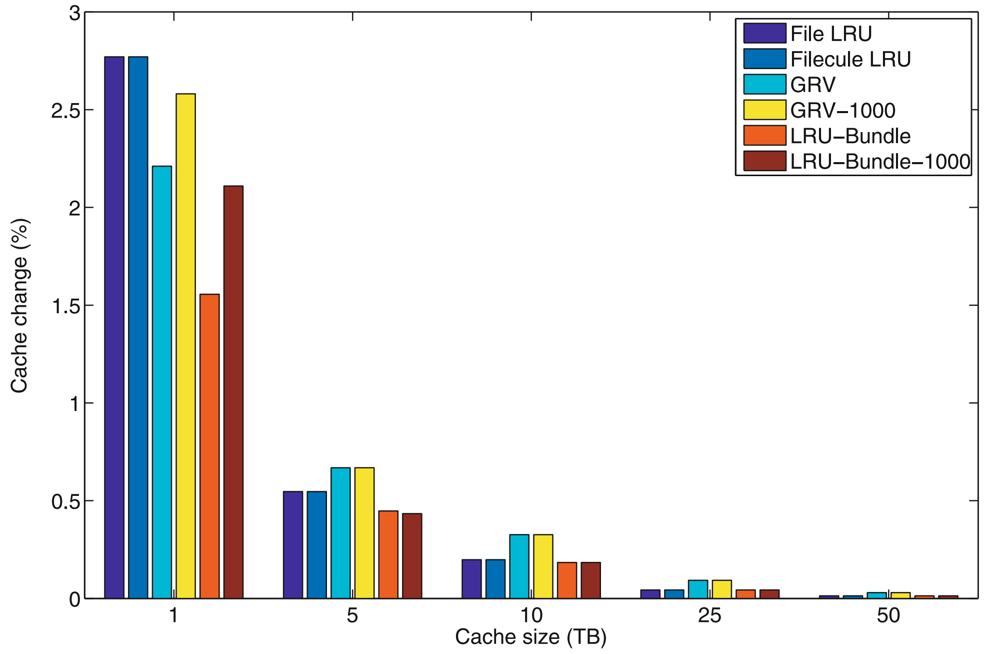


Fig. 13 Average percentage of cache change for different cache sizes

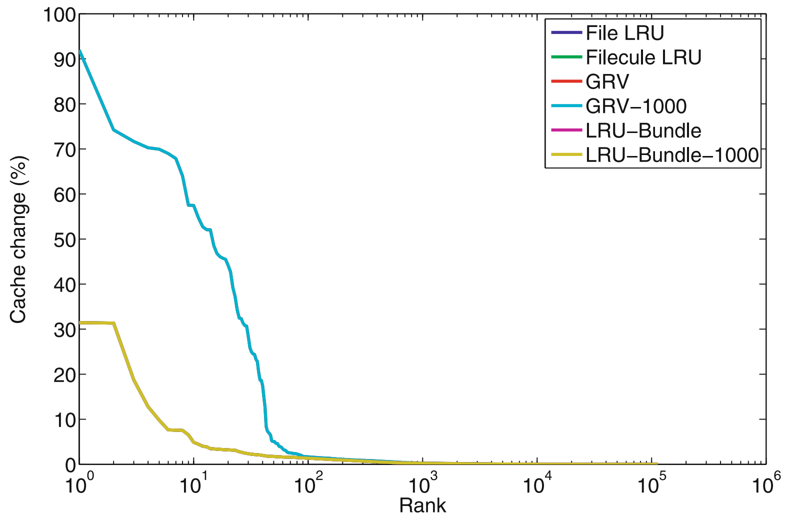


Fig. 14 Percentage of cache change for cache size of 50 TB. File LRU = Filecule LRU = LRU-Bundle = LRU-Bundle-1000 and GRV = GRV-1000

Only two distributions are visible because the curves match exactly for all LRU-based experiments and for all GRV-based experiments, respectively. Notice that even if the average percentage changes in cache per job are very close to each other (both groups of algorithms show under 0.1% cache percentage change, with the non-LRU-based algorithms higher, as shown in Fig. 13), the individual cache changes per job are significantly different.

5.2 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 exhibits high byte hit rate compared with the more elaborate GRV algorithm (Fig. 11). 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

Fig. 15 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).

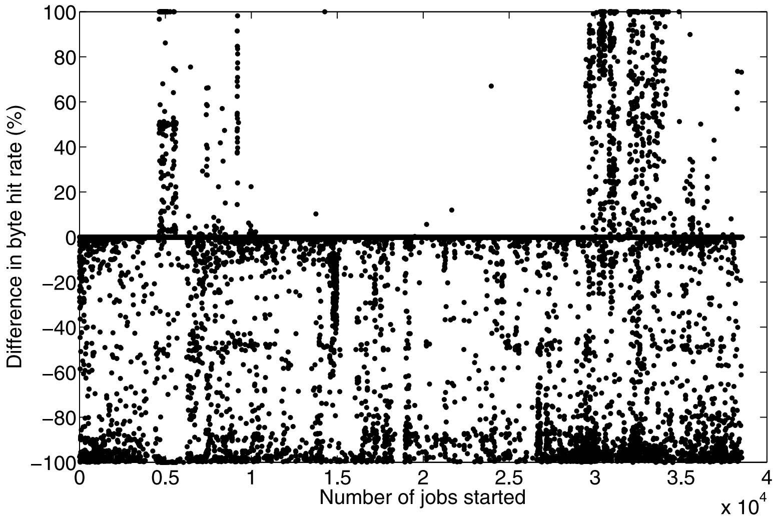
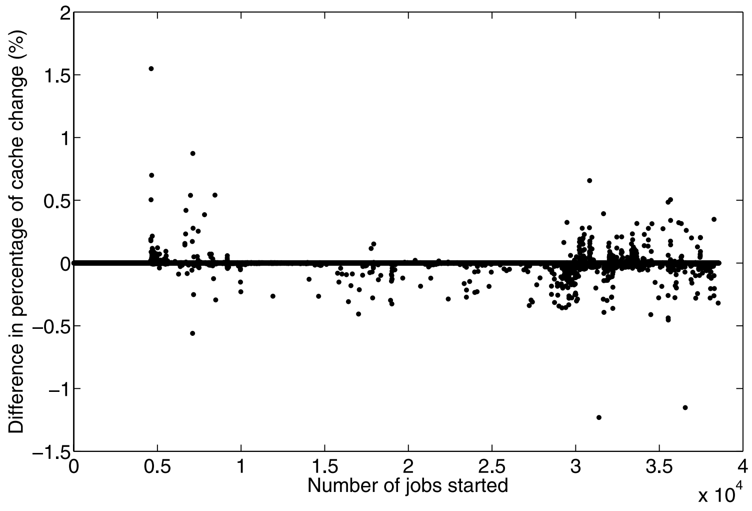


Fig. 16 Difference in percentage of cache change between filecule LRU with 1-month window and Filecule LRU with optimal filecules



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 (Fig. 15). 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 16 shows the difference between the percent-

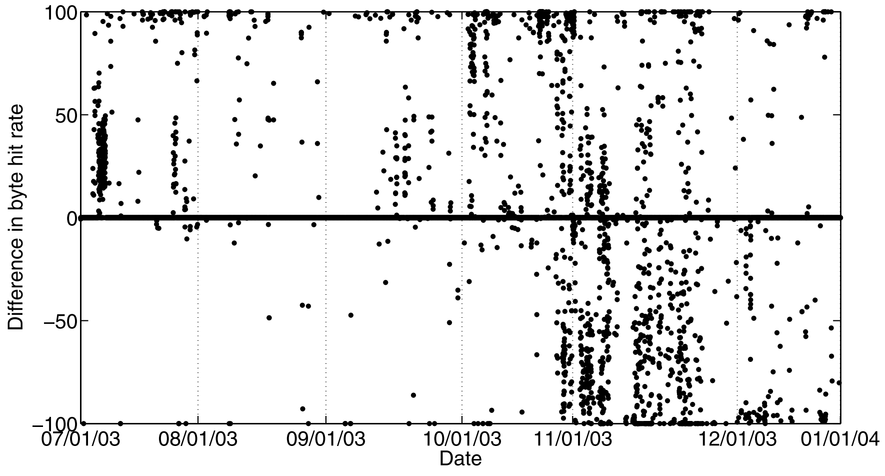


Fig. 17 Difference in byte hit rate between filecule LRU with 6-month window and 1-month window. 22,499 (92%) jobs have equal byte hit rates. Filecule LRU with 6-month window has lower byte hit rate for 1,005 jobs (4%). Filecule LRU with 1-month window higher byte hit rate for 888 jobs (3.9%)

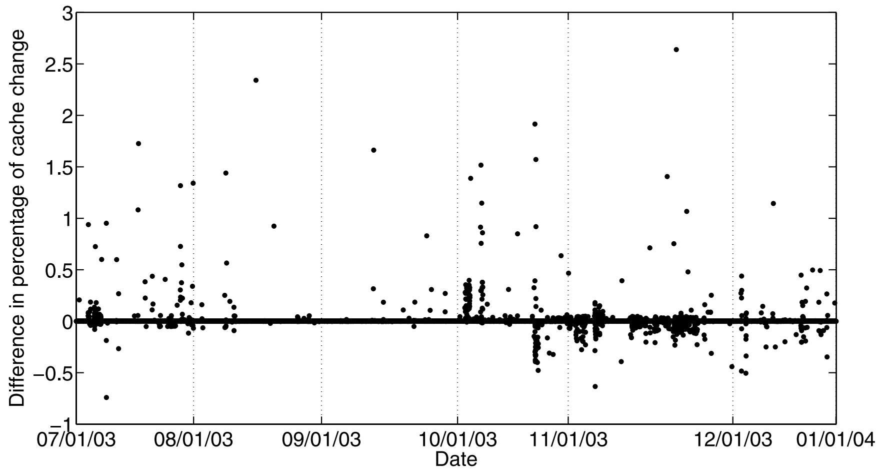


Fig. 18 Difference in percentage of cache change between filecule LRU with 6-month window and 1-month window

age 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 17 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 (Fig. 18).

To summarize, 1-month history is sufficient for determining filecules 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 [22] may lead to even better results.

5.3 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.

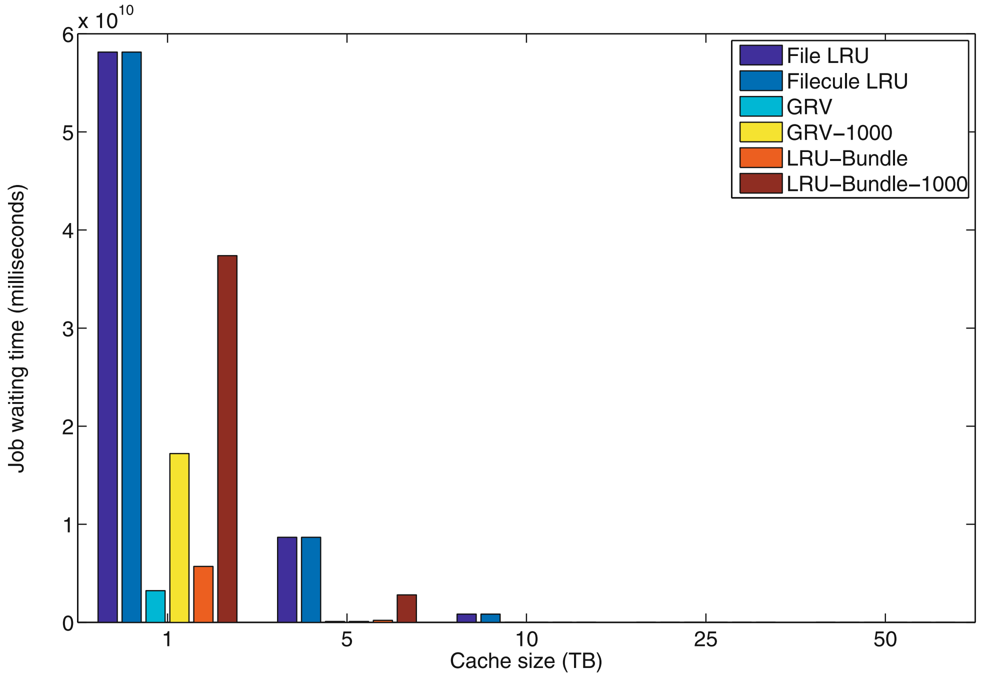


Fig. 19 Average job waiting time for different cache sizes

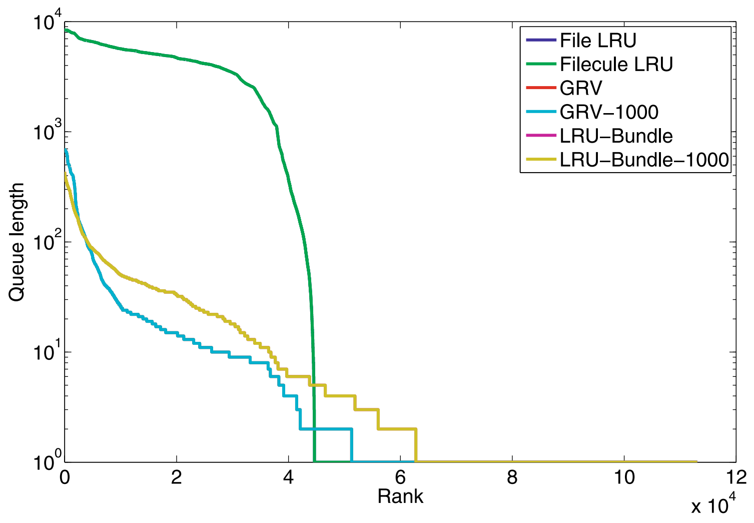


Fig. 20 Queue length for cache size of 10 TB. File LRU = Filecule LRU, GRV = GRV-1000 and LRU-Bundle = LRU-Bundle-1000

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

Similar to Fig. 14, Fig. 20 presents the lengths of the queue jobs per job in decreasing order for a cache size of 10 TB. As before, some of the plots overlap. A shorter queue is better and this is the case with the algorithms that use GRV-based job reordering (thus, GRV and LRU-Bundle).

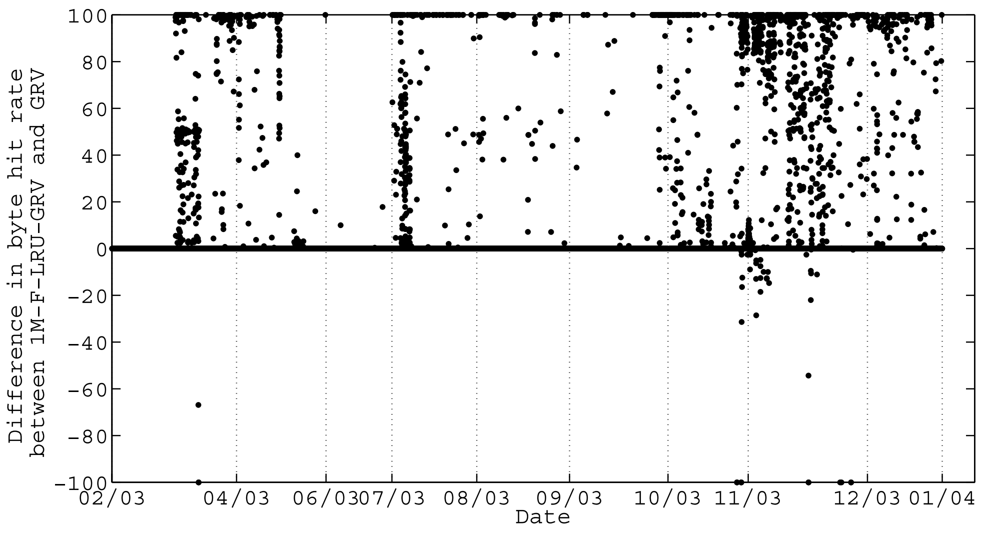


Fig. 21 Difference in byte hit rate of LRU-GRV with filecules and GRV

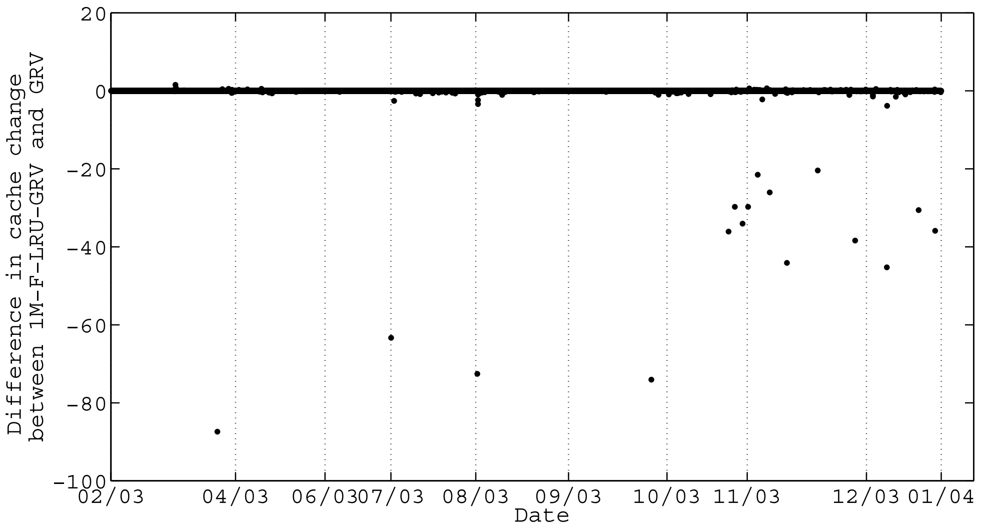


Fig. 22 Difference in percentage of cache change of LRU-GRV with filecules and GRV

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 GRV.

Figure 21 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 22 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 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.

6 Related work

This article builds upon significant previous work in systems research. In particular, we benefited from previously published work on workload characterization in file systems, the web, and science grids; data management in science grids; workload-aware file grouping and resource management.

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 [31] and availability [32]. Synthetic workloads have been proposed in [29].

The need for a better understanding of workloads and a set of realistic benchmarks has been presented in [25] in the context of correctly evaluating solutions for decentralized systems.

Web workloads have been studied intensively since the late 90s. Web client traces have been studied by Barford et al. [7] and were shown to follow a log-normal distribution with Pareto tails for web file sizes. Web server workloads have been studied extensively in [3, 5] and the Zipf distribution was confirmed for file popularity. One of these studies also showed a Pareto distribution of file sizes, result confirmed by another, earlier study [13]. Studies of traces from a web proxy within an ISP [6] confirmed the heavy-tailed file-size distribution and the Zipf-distribution of file popularity. In [9], Breslau et al. analyze web proxy cache traces from different sources. They show that the page request distribution follows a Zipf-like distribution. They also show that there is weak correlation between page size and popularity.

File systems characteristics have also been intensively studied. Douceur and Bolosky [17] analyzed the size and lifetime of Windows files in the file systems at Microsoft Corporation. The mean file sizes in these file systems varied from 64 KB to 128 KB, but the median file sizes was just 4 KB, indicating a large number of small files. The file size distribution was shown to be log-normal. The file lifetimes were observed to follow a hyper-exponential distribution. Similar observations were made by Vogels in [56].

More recently, Tanenbaum et al. [54] studied the file size distribution on Unix systems. They analyzed the file sizes on the Unix machines at the Computer Science department of the Vrije Universiteit during 2005 and showed that the median file sizes have doubled since 1984 [37]. The largest file (2 GB) is about 4,000 times bigger than the largest file in 1984.

More recently, the characterization of peer-to-peer systems brought new insights on the properties of (mainly) media files. A study on the popularity a files in Gnutella [50] found that the very popular files were equally popular while the rest of the files followed a Zipf-like distribution. A more nuanced study [22] shows the geographical variation of content popularity. Analyzes of file-sharing applications such as Gnutella, Kazaa and Napster confirm that different file size distributions emerge with different content types (predominantly multimedia in this case) [48].

Data management in science grids is one of the main foci for researchers and users of grid computing: scheduling data-intensive jobs [33, 44–46] and scheduling data transfers [2, 11] have received significant attention. iRODS [43] 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.

Caching in data grids has also been the focus of much effort. Results from Otoo, Shoshani et al. [39–41], presented earlier in this paper, show a variety of improvements in disk cache algorithms aimed at and tested in scientific data analysis environments.

Workload-aware resource management has been traditionally addressed at the system level in disk management [14, 51–53]. File grouping techniques have been investigated in the context of mobile computing [34], caching in distributed file systems [4], web proxies [49], disk access [20], and Microsoft Windows updates [23]. Solutions that implement application-aware data management include LOCKSS [36] for library archives and the Google File System [21] for Google data processing. In the peer-to-peer systems context, solutions that take advantage of observed patterns include file location mechanisms that exploit the relation between stored files [12], information dissemination techniques [26] that exploit overlapping user interests in data [28], and search algorithms [1] adapted to particular overlay topologies [47].

7 Summary and future work

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.

7.1 Impact on workload modeling

Synthetic workloads are often generated for supporting experimental efforts when appropriate 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 noticed 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.

Finally, an important contribution of our work is introducing filecules as a usage-based data abstraction for grouping files and evaluating its impact on data management. We present the characteristics of the filecules identified in the DZero traces and show that this way of grouping data does not follow either the size and popularity distributions that were revealed in many other distributed systems.

7.2 Impact on 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 (Sect. 5). 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.

7.3 Future work

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.

Another direction for future work is to apply filecules for data replication and placement, as discussed previously. Finally, we would like to verify the generality of the patterns we identified on other scientific workloads. Recent efforts led to the creation of a Grid Workload Archive [24] that may make available other relevant traces in the near future.

Looking further ahead, this work signals the need for expressive resource management policies that are able to specify relationships between files. Moreover, these relationships should then be interpreted by a resource management service capable of exploiting inter-file relationships. We are currently developing a flexible and general policy language for resource management along with a policy enforcement mechanism for large-scale, distributed systems.

Acknowledgements This material is based upon work partially supported by the National Science Foundation under Grants No. CNS-0716343 and CNS-0831785. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Adamic, L., Huberman, B., Lukose, R., Puniyani, A.: Search in power law networks. Phys. Rev. E **64**, 46135–46143 (2001)
2. Allen, M., Wolski, R.: The Livny and Plank-Beck problems: studies in data movement on the computational grid. In: Supercomputing, 2003
3. Almeida, V., Bestavros, A., Crovella, M., de Oliveira, A.: Characterizing reference locality in the WWW. In: 4th International Conference on Parallel and Distributed Information Systems, pp. 92–103, Dec. 1996
4. Amer, A., Long, D.D.E., Burns, R.C.: Group-based management of distributed file caches. In: ICDCS, 2002
5. Arlitt, M.F., Williamson, C.L.: Internet web servers: workload characterization and performance implications. IEEE/ACM Trans. Netw. **5**(5), 631–645 (1997)
6. Arlitt, M., Friedrich, R., Jin, T.: Workload characterization of a web proxy in a cable modem environment. SIGMETRICS Perform. Eval. Rev. **27**(2), 25–36 (1999)
7. Barford, P., Bestavros, A., Bradley, A., Crovella, M.: Changes in web client access patterns: characteristics and caching implications. Proc. World Wide Web **2**, 15–28 (1999)
8. Bestavros, A.: 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 Distributed Processing, Washington, DC, USA, 1995, p. 338. IEEE Computer Society, Los Alamitos (1995)
9. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-like distributions: evidence and implications. In: INFOCOM (1), pp. 126–134, 1999
10. Brun, R., Rademakers, F.: In: An Object Oriented Data Analysis Framework, 1996
11. Catalyurek, U., Kurc, T., Sadayappan, P., Saltz, J.: Scheduling file transfers for data-intensive jobs on heterogeneous clusters. In: Proceedings of the 13th European Conference on Parallel and Distributed Computing (Europar), 2007

12. Cohen, E., Fiat, A., Kaplan, H.: Associative search in peer to peer networks: harnessing latent semantics. In: Infocom, San Francisco, CA, 2003
13. Cunha, C., Bestavros, A., Crovella, M.: Characteristics of www client-based traces. Technical report, Boston, MA, USA, 1995
14. Ding, X., Jiang, S., Chen, F., Davis, K., Zhang, X.: Diskseen: exploiting disk layout and access history to enhance I/O prefetch. In: USENIX Annual Technical Conference, 2007
15. Doraimani, S.: Filecules: a new granularity for resource management in grids. Master's thesis, University of South Florida (2007)
16. Doraimani, S., Iamnitchi, A.: File grouping for scientific data management: lessons from experimenting with real traces. In: 17th ACM/IEEE Symposium on High Performance Distributed Computing (HPDC), June 2008
17. Douceur, J.R., Bolosky, W.J.: 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, New York, NY, USA, 1999, pp. 59–70. ACM Press, New York (1999)
18. The DZero Experiment. http://www-d0.fnal.gov
19. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: enabling scalable virtual organizations. In: Lecture Notes in Computer Science, vol. 2150, pp. 1–4. Springer, Berlin (2001)
20. Ganger, G.R., Kaashoek, M.F.: Embedded inodes and explicit grouping: exploiting disk bandwidth for small files. In: USENIX Annual Technical Conference, pp. 1–17, 1997
21. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. In: SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, 2003, pp. 29–43. ACM Press, New York (2003)
22. Gish, A.S., Shavitt, Y., Tankel, T.: Geographical statistics and characteristics of p2p query strings. In: IPTPS2007—Proceedings of the 6th International Workshop on Peer-to-Peer Systems, February 2007
23. Gkantsidis, C., Karagiannis, T., Vojnovic, M.: Planet scale software updates. In: SIGCOMM, pp. 423–434, 2006
24. The Grid Workloads Archive. http://gwa.ewi.tudelft.nl/
25. Haeberlen, A., Mislove, A., Post, A., Druschel, P.: Fallacies in evaluating decentralized systems. In: The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06), 2006
26. Iamnitchi, A., Foster, I.: Interest-aware information dissemination in small-world communities. In: 14th IEEE International Symposium on High Performance Distributed Computing (HPDC), July 2005
27. Iamnitchi, A., Doraimani, S., Garzoglio, G.: Filecules in high-energy physics: characteristics and impact on resource management. In: 15th IEEE International Symposium on High Performance Distributed Computing (HPDC), pp. 69–79, June 2006
28. Iamnitchi, A., Ripeanu, M., Foster, I.: Small-world file-sharing communities. In: Infocom, Hong Kong, China, 2004
29. Iosup, A., Epema, D.: 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), Washington, DC, USA, 2006, pp. 313–320. IEEE Computer Society, Los Alamitos (2006)
30. Iosup, A., Epema, D.H.J., Couvares, P., Karp, A., Livny, M.: Build-and-test workloads for grid middleware: problem, analysis, and applications. In: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07), Rio de Janeiro, Brazil, 2007, pp. 205–210. IEEE Computer Society, Los Alamitos (2007)
31. Iosup, A., Jan, M., Sonmez, O., Epema, D.H.J.: The characteristics and performance of groups of jobs in grids. In: EuroPar, 2007
32. Iosup, A., Jan, M., Sonmez, O., Epema, D.H.J.: On the dynamic resource availability in grids. In: Grid, 2007
33. Khanna, G., Vydyanathan, N., Catalyurek, U., Kurc, T., Krishnamoorthy, S., Sadayappan, P., Saltz, J.: 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
34. Kuenning, G.H., Popek, G.J.: Automated hoarding for mobile computers. In: SOSP '97: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, 1997, pp. 264–275. ACM Press, New York (1997)
35. Loebel-Carpenter, L., Lueking, L., Moore, C., Pordes, R., Trumbo, J., Veseli, S., Terekhov, I., Vranicar, M., White, S., White, V.: Sam and the particle physics data grid. In: Computing in High-Energy and Nuclear Physics, Beijing, China, 2001
36. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H., Baker, M.: The LOCKSS peer-to-peer digital preservation system. ACM Trans. Comput. Syst. **23**(1), 2–50 (2005)
37. Mullender, S.J., Tanenbaum, A.S.: Immediate files. Softw. Pract. Exper. **14**(4), 365–368 (1984)
38. Otoo, E., Olken, F., Shoshani, A.: Disk cache replacement algorithm for storage resource managers in data grids. In: Supercomputing '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Los Alamitos, CA, USA, 2002, pp. 1–15. IEEE Computer Society Press, Los Alamitos (2002)
39. Otoo, E., Rotem, D., Romosan, A.: Optimal file-bundle caching algorithms for data-grids. In: SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, Washington, DC, USA, 2004, p. 6. IEEE Computer Society, Los Alamitos (2004)
40. Otoo, E.J., Rotem, D., Romosan, A., Seshadri, S.: File caching in data intensive scientific applications on data-grids. In: Data Management in Grids, pp. 85–99, 2005
41. Otoo, E.J., Rotem, D., Seshadri, S.: Efficient algorithms for multi-file caching. In: 15th International Conference Database and Expert Systems Applications, pp. 707–719, 2004
42. Rajasekar, A., Wan, M., Moore, R., Kremenek, G., Guptil, T.: Data grids, collections, and grid bricks. In: MSS '03: Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), Washington, DC, USA, 2003, p. 2. IEEE Computer Society, Los Alamitos (2003)
43. Rajasekar, A., Wan, M., Moore, R., Schroeder, W.: A prototype rule-based distributed data management system. In: Workshop on Next Generation Distributed Data Management, 2006
44. Ranganathan, K., Foster, I.: 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
45. Ranganathan, K., Foster, I.: Identifying dynamic replication strategies for a high performance data Grid. In: International Workshop on Grid Computing, 2001
46. Ranganathan, K., Foster, I.: 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
47. Ripeanu, M., Iamnitchi, A., Foster, I.: Mapping the Gnutella network: properties of large-scale peer-to-peer systems and implications for system design. Internet Comput. **6**(1), 50–57 (2002)
48. Saroiu, S., Gummadi, K.P., Dunn, R.J., Gribble, S.D., Levy, H.M.: An analysis of internet content delivery systems. SIGOPS Oper. Syst. Rev. **36**(SI), 315–327 (2002)
49. Shriver, E.A.M., Gabber, E., Huang, L., Stein, C.A.: Storage management for web proxies. In: Proceedings of the General Track: 2002 USENIX Annual Technical Conference, Berkeley, CA, USA, 2001, pp. 203–216. USENIX Association, Berkeley (2001)
50. Sripanidkulchai, K.: The popularity of gnutella queries and its implications on scalability. White paper, http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html (2001)

51. Staelin, C., Garcia-Molina, H.: Clustering active disk data to improve disk performance. Technical Report CS–TR–298–90, Princeton, NJ, USA, 1990
52. Staelin, C., Garcia-Molina, H.: Smart filesystems. In: USENIX Winter, pp. 45–52, 1991
53. Tait, C.D., Duchamp, D.: Detection and exploitation of file working sets. In: Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS), Washington, DC, 1991, pp. 2–9. IEEE Computer Society, Los Alamitos (1991)
54. Tanenbaum, A.S., Herder, J.N., Bos, H.: File size distribution on Unix systems: then and now. SIGOPS Oper. Syst. Rev. **40**(1), 100–104 (2006)
55. Terekhov, I.: Meta-computing at d0. In: Nuclear Instruments and Methods in Physics Research, Section A, NIMA14225, vol. 502/2-3, pp. 402–406, 2002
56. Vogels, W.: File system usage in windows nt 4.0. In: SOSP '99: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles, New York, NY, USA, 1999, pp. 93–109. ACM Press, New York (1999)

Adriana Iamnitchi is Assistant Professor in the Computer Science and Engineering Department of University of South Florida. Adriana received her Ph.D. in Computer Science from the University of Chicago in 2003. Her research interests are in distributed systems with a focus on self-organization and decentralized control in large-scale Grid and peer-to-peer systems.

Shyamala Doraimani works as a software developer at Cogency Software, developing automated accounting and operations solutions for the alternative investment industry. She had experience in design and development of control system software for energy management at Siemens Power Transmission and Distribution. Shyamala received her master in computer science from University of South Florida in 2007, working on data access patterns for high energy physics applications.

Gabriele Garzoglio works as a computer professional at the Fermi National Accelerator Laboratory. He leads the Open Science Grid group, a team of Software Engineers that provides middleware solutions for large distributed systems. In recent years, Gabriele has focussed on data and workload management for high energy physics applications, as well as resource selection and authorization. Gabriele has a Laurea Degree in Physics from University of Genova, Italy (1996), and received his Ph.D. in Computer Science from DePaul University, Chicago, IL (2006).