Filecules in High-Energy Physics: Characteristics and Impact on Resource Management

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

Shyamala Doraimani
Computer Science and Engineering
University of South Florida
sdoraima@cse.usf.edu

Gabriele Garzoglio
Fermi National Laboratory
Batavia, IL
garzogli@fnal.gov

Abstract

Grid computing has reached the stage where deployments are mature and many collaborations run in production mode. Mature Grid deployments offer the opportunity for revisiting and perhaps updating traditional beliefs related to workload models, which in turn leads to the re-evaluation of traditional resource management techniques.

This paper analyzes usage patterns in a typical Grid community, a large-scale data-intensive scientific collaboration in high-energy physics. We focus mainly on data usage, since data is the major resource for this class of applications. Our observations led us to propose a new abstraction for resource management in scientific data analysis applications: we define a filecule as a group of files that is always used together. We show that filecules exist and present their characteristics. The existence of filecules suggests a new granularity for data management, which, if incorporated in design, can significantly outperform the traditional solutions for data caching, replication and placement based on single-file granularity. We reason about the impact of filecules on resource management and show compelling evidence for using this abstraction when designing data management services.

1 Introduction

Sustained effort is ongoing to support various scientific communities and their large-scale data-sharing and data-analysis needs through a distributed, transparent infrastructure (such as for GriPhyN [7], PPDG [28], and other projects). This infrastructure necessarily includes components for file location and management as well as for computation and data transfer scheduling. However, there is little information available on the specific usage patterns that emerge in these data-intensive, scientific projects.

This paper analyzes the characteristics of a production-mode data-intensive high-energy physics collaboration, the DZero Experiment [12], hosted at Fermi National Accelerator Laboratory (FermiLab). In Grid terminology [17], the DZero Experiment is a virtual organization consisting of hundreds of physicists in 70+ institutions from 18 countries. Its purpose is to provide a worldwide system of shareable computing and storage resources that can together solve the common problem of extracting physics results from several Petabytes of measured and simulated data. In this system, data files are read-only and the typical jobs analyze and produce new, processed data files. Tracing system utilization is possible via a software layer (SAM [24, 33]) that provides centralized file-based data management.

We analyzed logs from January 2003 to May 2005, amounting to about 234,000 job runs submitted by 561 users from 34 different Internet domains in 11 countries on three continents. We have detailed data access information about half of the jobs: these 115,895 jobs involve more than 13 million accesses to about 1.13 million distinct files (see Table 1 for the exact figures). Jobs are run on multiple files, on average 108 files per job. Figure 1 shows the distribution of the number of files per job.

Our analysis showed an important pattern: scientific data usage translates into requests for groups of correlated files. This fact suggests a new granularity for data management, which can be exploited to design solutions that may significantly outperform the traditional solutions based on single-file granularity. We experiment with one data-management

Table 1. Characteristics of traces analyzed per data tier.

Data Tier	Users	Jobs	Files	Input/Job (MB)	Time/Job (hours)
Reconstructed	320	17898	515677	36371	11.01
Root-tuple	63	1307	60719	83041	13.68
Thumbnail	449	94625	428610	53619	4.89
Others	435	120962	N/A	N/A	7.68
All	561	233792	N/A	N/A	6.87

service, caching, and observe that by applying traditional caching strategies to filecules instead of files we obtain significantly better performance: a 5-fold increase in hit rate.

While it is well acknowledged that scientific applications often process multiple input files [4, 25, 26, 27], to the best of our knowledge our study is one of the first quantitative analysis that focus on the size of the input data in scientific workloads. Other characterizations of real workloads typical of Grid communities focused on batch-pipelined scientific workloads [34].

We also evaluate the feasibility of applying known peer-to-peer strategies—such as BitTorrent—for efficient and low-cost data transfers based on real usage patterns. We observed that while the size of the data to be transfered may indeed justify parallel downloads from multiple sources as in BitTorrent, the relatively small number of concurrent users and sites that make use of a particular set of data does not call for using such a strategy.

In addition, we discuss the effects of filecules on other resource management services, such as data replication and replica placement algorithms, data transfer and job scheduling.

Section 2 gives a brief overview of the high-energy physics collaboration that provided the traces analyzed in this paper. In Section 3 we formally define filecules and present their characteristics in terms of size and popularity. Section 4 is a first proof that managing data at the granularity of filecules has benefits over the traditional single-file granularity. Section 5 verifies whether using BitTorrent for scheduling data transfers is appropriate in this environment. Finally, we conclude with a discussion of the consequences for resource management that our study highlights in Section 6 and summarize our results and plans for future work in Section 8.

2 The DZero Experiment: A High-Energy Physics Collaboration

DZero is one of the two experiments currently processing data from the Tevatron collider at Fermilab. DZero 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 1GB in size. Every bit of raw data is accessed for further processing/filtering. Data derived from pre-processing and filtering is then classified based on the physics events they represent. This section discusses the typical applications that operate on high energy physics data, the computing infrastructure of DZero, and the format of the traces studied in this paper.

2.1 Computation for High-Energy Physics

Modern high-energy physics experiments, such as DZero, typically acquire more than one TB of data per day and move up to ten times as much. To give an example, during the past year, the SAM system has stored more than half a petabyte of data for DZero. Aside from the stream of data from the detector, various other computing activities contribute to the one terabyte 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, spin, and others. The original format is instead very closely dependent on the hardware layout of the detector, in order to guarantee the performance of the data acquisition system, and is not suitable for data analysis.

The production of simulated events, also called montecarlo production, is necessary for understanding and isolating detector characteristics related to hardware, such as particle detection efficiency, or to physics phenomena, such as signal to background discrimination.

Finally, data analysis mainly consists of the selection and statistical study of particles with certain characteristics, with the goal of achieving physics measurements.

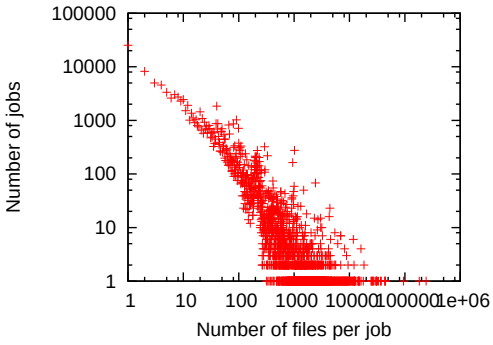

Figure 1. The number of input files per job.

2.2 Computing Infrastructure in DZero

The DZero experiment relies on the SAM middleware [33] for its data handling needs. 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 to 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. Because of 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 analyzed in this paper, 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 [9] and are generally input to analysis applications.

For the data handling middleware, an application running on a dataset defines a job or "project". Projects are initiated by a user on behalf of a physics group and typically trigger data movement.

2.3 Traces

The studies presented hereby utilize data from the SAM data processing history database between January 2003 and March 2005. Two types of traces have been selected for our studies: file traces and application traces.

File traces show what files have been requested with every job run during the period under study. These traces are used to study the presence of filecules in the DZero computing activity.

Application traces, instead, list summary information about jobs. The information includes metadata for the application (application name, version, and family), for the dataset processed (data tier), as well as general data, such as the user name and group that initiated the job and the location (node name) and start/stop time of the job.

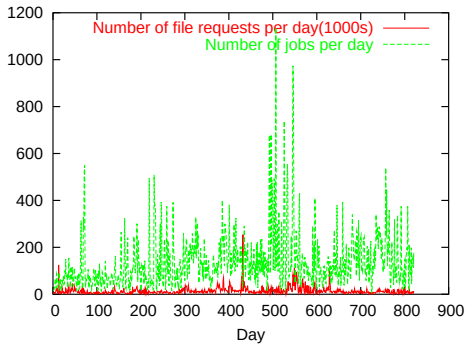

Figure 2. The number of jobs and the number of file requests (in '000s) per day.

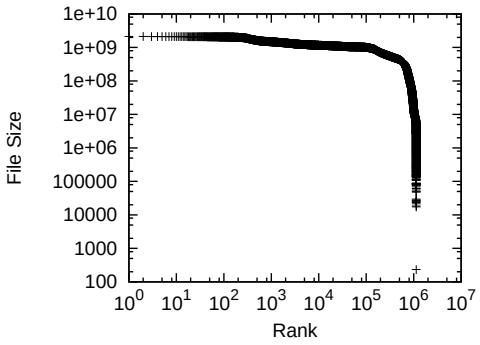

Figure 3. File size distribution.

3 Filecules

Understanding user behavior by analyzing traces from real user communities has multiple benefits. First, it quan-

Table 2. Characteristics of analyzed traces per location.

Domain	Jobs	Submission nodes	Sites	# users	# filecules	# files	Total data (GB)
.gov	3319711	12	1	466	95234	945031	4930850
.de	390186	5	4	23	33403	100257	268815
.uk	131760	8	4	21	23876	62427	117097
.edu	54672	18	12	32	14504	36868	41081
.cz	7400	1	1	1	4789	7660	9869
.ca	5719	5	2	4	649	8937	22341
.fr	5086	2	1	11	1767	18215	23958
.nl	3854	3	2	8	888	38812	44012
.mx	146	1	1	1	32	1589	349
.br	12	2	2	2	2	2	2
.cn	4	1	1	2	2	62	31
.in	3	1	1	2	2	2	0.70

tifies resource requirements and allows for better resource provisioning. One of the strongest examples in this class is the Zipf distribution of web requests and its implications on caching [8]. Other examples include the quantification of free riding in Gnutella [3] or the amount of traffic generated by specific user communities in peer-to-peer activities [31, 30, 29, 23].

Second, it provides a credible workload for evaluating new solutions via simulations or emulations. A notable example in this context is the study of the Internet topology [15] that corrected the modeling of the Internet from a random to a power-law graph.

Third, understanding usage patterns can lead to new ways of improving the overall system performance. Solutions that took advantage of observed patterns include file location mechanisms that exploit the relation between stored files [11], information dissemination techniques [21] that exploit overlapping user interests in data [22], and search algorithms [2] adapted to particular overlay topologies [29].

In this study we analyze the use of data while aiming to get insight into what data management techniques are better suited for this class of data-intensive applications. One characteristic of many scientific domains is the processing of large amounts of data organized in multiple files (datasets or collections) that form the input set for various jobs. Such collections of files may consist of multiple smaller subsets of interdependent files that we call filecules.

In particular, we aim to understand the effectiveness of using filecules as a new abstraction for designing and implementing data management techniques. To this end we look at particular characteristics that influence data management decisions and performance, such as size (Figures 6 and 7) and popularity distribution of filecules (Figure 8), the degree of filecule sharing among various users (Figure 4), the number of filecules per job (Figure 5), and the number of jobs submitted to computational resources located at different sites in the DZero collaboration (Table 2).

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 (instead of multiple-file filecules and single files).

Formally, a set of files $F_1, ..., F_n$ form a filecule G if and only if $\forall F_i, F_j \in G$ and $\forall G'$ such that $F_i \in G'$, then $F_j \in G'$. Some properties result directly from this definition:

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.

Table 2 presents some statistics on the number of filecules and access to data from various groups in DZero. It can also be observed the distribution of user activity (number of jobs submitted from a particular domain) with the DZero's location being by far the most active.

Figure 4 presents the number of users who access the same filecule over the period of our study. While about 10% of the filecules are accessed by one user only, a significant fraction of filecules have a larger user population, capped at 44. Our studies revealed no correlation between filecule popularity and filecule size.

It is relevant to note that, due to lack of information in traces, we cannot distinguish between the popularity of

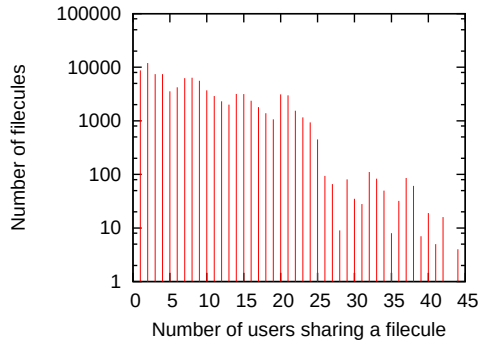

Figure 4. Number of users sharing a filecule.

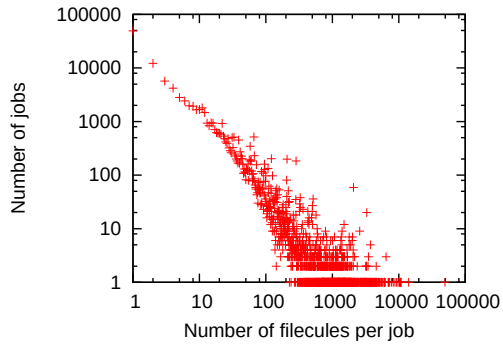

Figure 5. Number of filecules per job.

physics events recorded in a file and the popularity of that file. Because in the experiments whose traces we analyze there is no event index available, processing a data file means unpacking and looking at each event in the file. In a previous setup of the DZero experiments, data files contained an event table, which allowed random access to events in a file. Because performance and maintain costs this structure was not maintained for the more recent experiments that we analyze.

3.1 Size of Files and Filecules

The model traditionally accepted for file size distribution is heavy tailed. This model was built based on empirical evidence from file systems [14] and web pages [6]. More recent studies of peer-to-peer 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) [30]. We observed that scientific data reflect different file size distribution (Figure 3). Perhaps more important than the distribution itself is the observation that other rules govern the sizes of scientific data: First, some characteristics are domain-dependent. For example, in DZero an event as recorded from the accelerator is about 250 KB and a raw file is a sequence of such events. Second, deployment specific decisions, such as limits on the size of a file, may dictate file size distribution: in DZero, raw data is maintained in 1GB files.

Figures 6 and 7 present the size of filecules grouped by the data tier to which they belong.

3.2 Popularity

The distribution of filecule popularity showed by our studies (Figure 8) does not follow the traditional Zipf distribution model. The same phenomenon was revealed in the study of peer-to-peer networks [30], but the explanation for this behavior cannot be transferred between the two systems: while users of music-sharing networks do not repeatedly ask for the same file (which reduces the overall popularity of a file, and ”flattens” the distribution), scientists repeatedly request the same file for different computations. A more accurate explanation may be the role of geographical locality in user interests: the data space may be inherently partitioned among geographically remote groups who focus on disjoint parts of the data space.

A rigorous interpretation of data popularity distribution and its correlation with geographical locality may lead to improved designs for cache replacement techniques, data replication, and replica placement. Unfortunately, the geographical information contained in the DZero traces cannot be reliably mapped to user location, as it is more a metric of preferred computing pools.

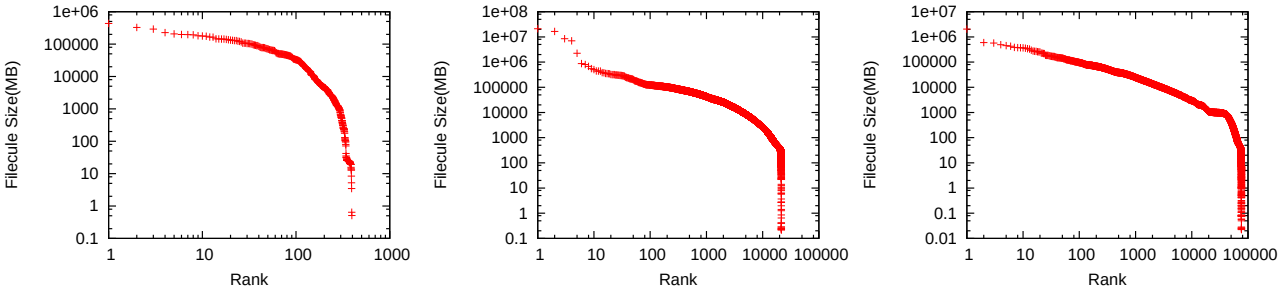

Figure 6. Size of filecules (in MB) per data tier. Left: roottuple tier. Center: reconstructed tier. Right: thumbnail tier.

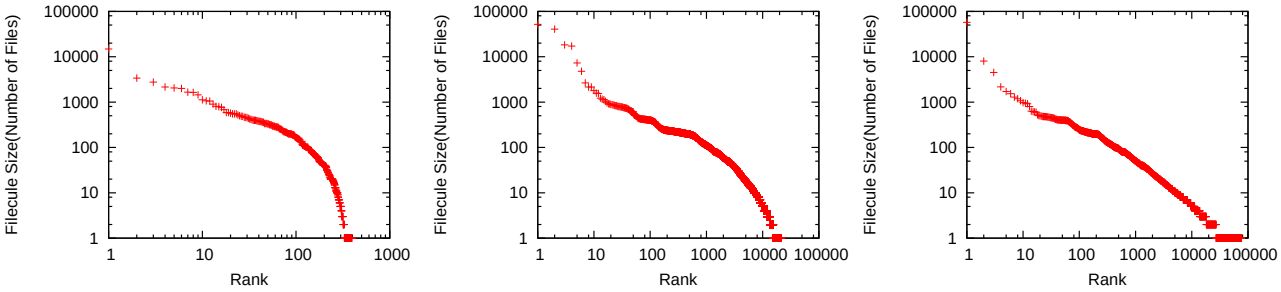

Figure 7. Number of files per filecule. Left: roottuple. Center: reconstructed. Right: thumbnail.

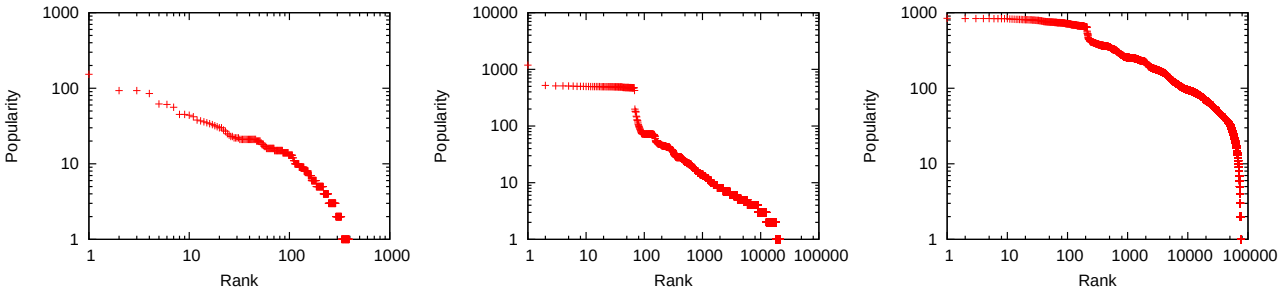

Figure 8. Popularity distribution (number of requests) for filecules per data tier. Left: roottuple. Center: reconstructed. Right: thumbnail.

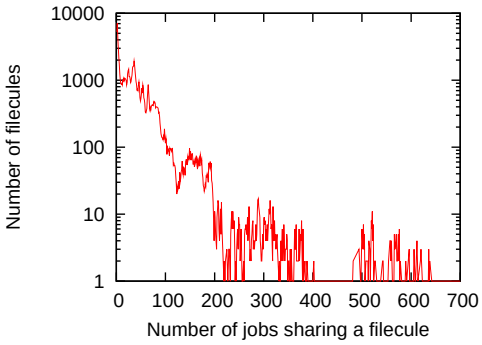

Figure 9. Number of requests per filecule.

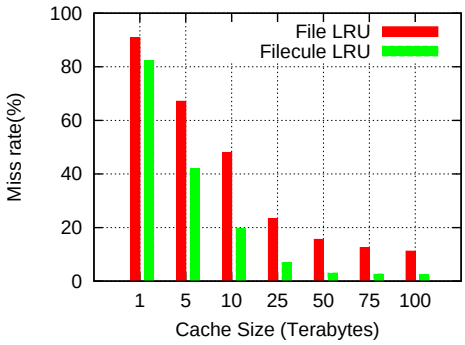

Figure 10. Miss rate for LRU cache replacement algorithm for file vs. filecule granularity.

Figure 9 presents the number of requests per filecule for the entire set of traces. It shows that while thousands of filecules are requested fewer than 50 times, there are tens of filecules that are requested more than 300 times during the interval of our study.

4 Cache Replacement Policy: Filecules vs. Files

For an initial evaluation of the advantages of using filecules vs. files we experimented with a well-known and very commonly used technique for data management, caching. A cache in this context is a large storage space used to store data transferred from another site or from tapes. We used the least recently used (LRU) algorithm for data replacement because of its simplicity and because of its use at FermiLab.

In LRU, to make room for more data, the file with the oldest timestamp (that is, the least recently used) is evicted. We compare via simulations the performance of the LRU cache replacement algorithm at file and filecule granularity. That is, for filecule LRU, we load the entire filecule of which a requested file is member and evict the least recently used filecules to make room for it.

We ran experiments for 7 different cache sizes (Figure 10) between 1TB and 100 TB. These are reasonable sizes for the context we are addressing: for example, disk caches vary from 1GB to 5 TB in DZero, are up to 150 TB [18] in CDF [10] and about 70 TB [18] in DESY [13]. The largest filecule in our experiments is 17TB.

Filecule LRU clearly outperforms file LRU, as can be seen in Figure 10: the miss rate for filecule LRU is significantly lower (up to 4 to 5 times for large cache sizes) than the miss rates for file LRU. However, it is important to note that the difference in performance is relatively small (about 9.5%) for cache size of 1 TB.

Otoo et al observe in [25] that cache replacement policies based solely on file popularity are not efficient for environments where multiple files are requested simultaneously. They propose a file eviction algorithm that considers, in addition to file popularity, the membership to a bundle and the size of the bundle. This strategy does not require the identification of filecules. We leave as future work the comparison of this strategy with filecule LRU on the DZero traces.

5 Using BitTorrent for Filecule Distribution

A question often raised in the research community is how successfully can solutions from the peer-to-peer domain be applied to Grid computing. Common to the two subdomains of distributed systems is the emphasis on access to data [16]. In particular, file location and file transfer are basic services in both communities.

A peer-to-peer solution of particular interest for the transfer of large files is BitTorrent. Developed and released for public use in 2001, BitTorrent [1] is a collaborative file-sharing protocol that is designed to work efficiently under flash crowd conditions where large numbers of users are simultaneously attempting to access and retrieve the same file. BitTorrent users make available chunks of the file to other peers while downloading the missing chunks from other BitTorrent clients. This mechanism reduces the load on peers that have a complete copy of the file by enabling a peer to download chunks from many peers. In addition, by using an exchange mechanism it enforces fair sharing.

BitTorrent is highly successful and is widely used in distributing large (multi-gigabyte) files. As the number of peers involved in the transfer of one file increases, the download time remains constant, demonstrating excellent scalability. It thus became a potentially interesting tool for scheduling data transfers in the context of desktop Grids [36, 35]. However, while effort is channeled toward adapting BitTorrent for the scientific community, to our knowl-

edge there is no study that attests the feasibility of BitTorrent from the perspective of real usage behavior.

The question we answer in this section is: Given the patterns of the DZero collaboration, would a mechanism like BitTorrent be useful? In particular, are there enough users who simultaneously use/request the same data? Note that this question can be posed for any data granularity (i.e., individual files or filecules).

To answer this question, we focus on a small set of filecules with larger numbers of users (see Figure 4) and study the overlapping of usage time. We show the results for one such filecule in Figures 11 and 12. The filecule we present consists of 2 files with a total of 2.2 GB, it is accessed by 42 users from 6 sites in a total of 634 jobs. The largest group of users comes from FermiLab (38 users with 529 job submissions), followed by Germany with 3 users and 66 jobs.

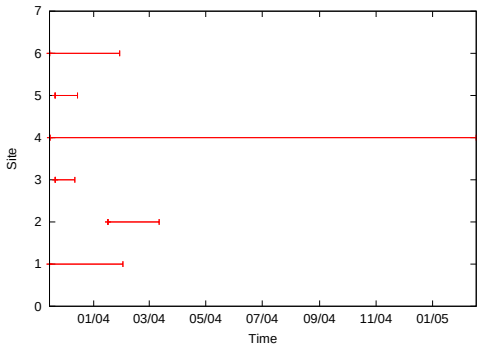

Figure 11. The time intervals in which a filecule is accessed from various sites.

In Figure 11 each horizontal line corresponds to the interval between the first and the last request for the filecule considered submitted per site. Because in scientific collaborations users are typically members of institutions, it is likely that users of the same institution will have access to a common local data storage. This is why in Figure 11 we consider a site as being one entity, despite the potentially large number of users at the site who might access data. The small number of simultaneous accesses to data does not plead for using BitTorrent in this particular context.

In Figure 12 we disassociate the users from their locations in an attempt to understand whether a different mapping of users to institutions would make BitTorrent a more appealing solution. Figure 12 thus presents the time interval between the first and last requests for the same filecule per user. While more activity is visible (there are periods when 10 users might store at least partial copies of the filecule and thus could provide data to another user), the load would hardly justify the use of BitTorrent for data transfers.

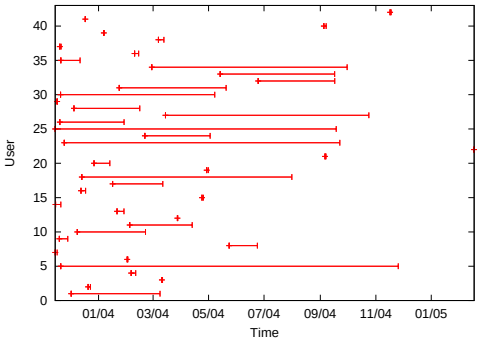

Figure 12. The time intervals in which a filecule is accessed by users.

However, note that the intervals presented are in fact not continuous: the assumption made in this analysis is that filecules are stored for the entire period of their use, which is an optimistic assumption. However, we want to stress that this result applies to the particular traces we study, traces that depend on the characteristics of the scientific computations and data and may not be portable to a different scientific community.

6 Consequences for Resource Management

We showed that filecules are a useful abstraction for data caching, but the potential for efficient data management is not limited to caching. For example, scheduling data transfers while accounting for filecules can lead to significant improvements.

Proactive data replication is one of the main motivations for this work and filecules seem to provide the appropriate abstraction. The question ”What files to replicate?” can be answered based on considerations related to popularity, replication costs, but also membership to filecules and the status of the filecule (partially or not-replicated) on the destination storage. Investigating proactive data replication strategies based on filecules is therefore a relevant direction to explore.

Our preliminary study identified filecules based on global knowledge of the long history of the DZero traces and proved that information about filecules is beneficial for cache management. However, at the basis of all potential improvements in data management techniques based on filecules is the very problem of identifying them. While our study relied on an off line analysis of traces, in a real setting we need an infrastructure capable to adaptively and dynamically identify filecules.

If the file requests are centralized at, for example, a metascheduler, then the problem is trivially solved by main-

taining reasonably long logs and processing them. The more interesting problem arises, though, if there are no central points of collecting file requests or job submissions, as required for a scalable and reliable architecture.

One solution could be to use the "concentration" points that already exist in the system. For example, schedulers that collect job requests from different sites/groups of users. The problem in this case might be that such concentrators represent a relatively small user population and they might thus collect insufficient information to accurately identify filecules. Assessing the costs of filecule-aware data replication under inaccurate identification of filecules is a relevant problem in this context. Because inaccurately identified filecules can only be larger than the filecules detected using global knowledge, we expect higher replication costs in terms of used storage and transfer costs.

Indeed, our preliminary experiments with this scenario show that larger filecules are identified when only a part of the jobs submitted, and thus datasets requested, are considered. For these experiments we considered each site collects its own job submissions and shares no information with other sites. In this case, not surprisingly, the number of filecules correctly identified depends on the user activity at that site: the more job submissions, the more likely that the filecules will be smaller and thus more accurate. Note that without global information, identified filecules can only be larger than real filecules. Also note that the filecules identified from only partial information are correct from the perspective of the local user base.

7 Related Work

Previous work in identifying relationships between files has been done in the context of web caching and file systems. Different methods of grouping files based on file relationships have been proposed [5, 19, 20]. Amer et al. [5] propose a method to group files based on successor relationship identified from a sequence of accesses. Upon the request of a file, its entire group is retrieved. Ganger and Kaashoek [19] use explicit grouping in which files that are used one after the other are placed in adjacent locations on the disk and accessed as a whole group. Griffioen and Appleton [20] consider two files related (and thus, part of the same group) if they are opened within a specified number of file open operations from each other.

All previous solutions use access sequences to identify file relationships and exploit these relationships for prefetching. Indeed, experimental results show lower cache miss rates when groups of files are prefetched based on file relationships [5, 19, 20]. As in some previous solutions, filecules are multiple-file implicit groups, but, at least in the context of this article, they are more stable than other grouping definitions. For example, in [5, 19, 20], two files are grouped together if and only if all intermediate accesses between them remain the same. In contrast, files in a filecule are related irrespective of intermediate file accesses and the time of access between them.

In [32] Tait and Duchamp analyze the use of file working sets for improving cache performance using prefetching. Their algorithm builds distinct "working" trees based on file access sequence and patterns. For every job, they track the file access sequence and compare it with the existing working trees. Prefetching is delayed until the sequence matches only one working tree. When a unique working tree is identified, the remaining files of that working tree are prefetched. Their experiments with file access traces from a SunOS machine prove that LRU with prefetching outperforms conventional LRU.

In addition to prefetching, file groups have been used to improve cache replacement algorithms by identifying a more efficient grouping technique. In the context of scientific applications, Otoo et al. propose a file-bundle algorithm for cache replacement [26, 27, 25]. Their algorithm maximizes the job throughput by optimizing the set of files to maintain in cache. Given a queue of requests and an available cache size, their algorithm identifies the optimal set of files, according to some cost function, that fit in the available cache. This optimal set is called a file bundle. Experiments with synthetic workload and file size distributions (with file sizes varying between 1MB and 10% of the cache size) show that the resulting byte miss ratio is significantly lower than that of the modified Landlord algorithm proposed in [37].

8 Summary and Future Work

We analyzed traces from a relatively large high-energy collaboration focusing on aspects related to data usage. We propose a new abstraction for data management, namely filecules, and show that it is more efficient for a common caching technique that the traditional one-file data granularity. We emphasize that this is just a preliminary study that gave us confidence that more work is worth investing into this direction.

This paper invites a whole set of new questions. A question is how prevalent filecules are in other sciences with computational problems appropriate for the Grid. The computational requirements of other disciplines that are early Grid adopters, such as astronomy, medicine, astrophysics, or biology make us believe that many of our findings will be transferable to a larger class of applications.

Another question is related to more thorough characterization of filecules: How dynamic are they? Do files stay in the same filecules or do they change over time? An experiment that would be perhaps relevant for this questions is to analyze filecules formed at different times: are two filecules

that contain the same file identical?

Finally, an important set of questions is related to the design of data management techniques that could benefit from the filecule abstraction. We plan to design and carefully investigate the costs and benefits of filecule-aware cache replacement policies and to compare them with related ideas.

References

[1] Bittorrent, http://www.bittorrent.com.

[2] L. Adamic, B. Huberman, R. Lukose, and A. Puniyani. Search in power law networks. *Physical Review. E*, 64:46135–46143, 2001.

[3] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), 2000.

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

[5] A. Amer, D. D. E. Long, and R. C. Burns. Group-based management of distributed file caches. In *ICDCS*, pages 525–, 2002.

[6] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a web proxy in a cable modem environment. Technical Report HPL-1999-48, Hwelett-Packard Laroratories, 1999.

[7] P. Avery and I. Foster. The griphyn project: Towards petascale virtual data Grids. Technical Report GriPhyN-2001-15, 2001.

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *InfoCom*, New York, NY, 1999. IEEE Press.

[9] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework, 1996.

[10] The collider detector at fermilab, http://www-cdf.fnal.gov/physics/public/public.html.

[11] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Infocom*, San Fancisco, CA, 2003.

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

[13] Deutches Elektronen-Synchrotron, http://www.desy.de.

[14] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 59 – 70, Atlanta, Georgia, USA, 1999. ACM Press New York, NY, USA.

[15] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[16] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and Grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, 2003.

[17] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[18] P. Fuhrmann. dCache: the commodity cache. In *Twelfth NASA Goddard and Twenty First IEEE Conference on Mass Storage Systems and Technologies*, Washington DC, Spring 2004.

[19] 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.

[20] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *USENIX Summer*, pages 197–207, 1994.

[21] A. Iamnitchi and I. Foster. Interest-aware information dissemination in small-world communities. 2005.

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

[23] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa network. In *Workshop on Internet Applications*, San Francisco, CA, 2003.

[24] 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 *Computing in High-Energy and Nuclear Physics*, Beijing, China, 2001.

[25] 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.

[26] E. J. Otoo, D. Rotem, A. Romosan, and S. Seshadri. File caching in data intensive scientific applications on data-grids. In *Data Management in Grids*, 2005.

[27] E. J. Otoo, D. Rotem, and S. Seshadri. Efficient algorithms for multi-file caching. In *15th International Conference Database and Expert Systems Applications*, pages 707–719, 2004.

[28] Particle physics data grid project, http://www.ppdg.net.

[29] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Internet Computing*, 6(1):50–57, 2002.

[30] S. Saroiu, P. K. Gummadi, R. Dunn, S. D. Gribble, and H. Levy. An analysis of internet content delivery systems. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, USA, 2002.

[31] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, 2002.

[32] 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.

[33] I. Terekhov. Meta-computing at D0. In *Nuclear Instruments and Methods in Physics Research, Section A, NIMA14225*, volume 502/2-3, pages 402–406, 2002.

[34] D. Thain, J. Bent, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Pipeline and batch sharing in grid workloads. In *Proceedings of the Twelfth IEEE Symposum on High Performance Distributed Computing*, pages 152–161, Seattle, WA, June 2003.

[35] B. Wei, G. Fedak, and F. Cappello. Collaborative data distribution with BitTorrent for computational desktop grids, 2005.

[36] B. Wei, G. Fedak, and F. Cappello. Scheduling independent tasks sharing large data distributed with BitTorrent, 2005.

[37] N. E. Young. On-line file caching. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 82–86, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.