A Study on Data Deduplication in HPC Storage Systems

Dirk Meister, Jürgen Kaiser, Andre Brinkmann
Johannes Gutenberg-University, Mainz
Germany
{dirkmeister, j.kaiser, brinkmann}@uni-mainz.de

Toni Cortes
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
Spain
toni.cortes@bsc.es

Michael Kuhn, Julian Kunkel
University of Hamburg,
Germany
{kuhn, kunkel}@uni-hamburg.de

***Abstract*—Deduplication is a storage saving technique that is highly successful in enterprise backup environments. On a file system, a single data block might be stored multiple times across different files, for example, multiple versions of a file might exist that are mostly identical. With deduplication, this data replication is localized and redundancy is removed – by storing data just once, all files that use identical regions refer to the same unique data. The most common approach splits file data into chunks and calculates a cryptographic fingerprint for each chunk. By checking if the fingerprint has already been stored, a chunk is classified as redundant or unique. Only unique chunks are stored.**

This paper presents the first study on the potential of data deduplication in HPC centers, which belong to the most demanding storage producers. We have quantitatively assessed this potential for capacity reduction for 4 data centers (BSC, DKRZ, RENCI, RWTH). In contrast to previous deduplication studies focusing mostly on backup data, we have analyzed over one PB (1212 TB) of online file system data. The evaluation shows that typically 20% to 30% of this online data can be removed by applying data deduplication techniques, peaking up to 70% for some data sets. This reduction can only be achieved by a subfile deduplication approach, while approaches based on whole-file comparisons only lead to small capacity savings.

I. Introduction

Deduplication is a storage saving technique that is highly successful in enterprise storage. By removing duplicates within and across files, this general concept has been successfully applied to backup, virtual machine storage, and WAN replication [1]–[4]. The most common approach splits file data into chunks, using sophisticated chunking strategies, and calculates a cryptographic fingerprint for each chunk. Each chunk is then classified as redundant or unique by checking if the fingerprint has already been stored. Only data of unique chunks is stored.

This paper investigates the potential for data deduplication in HPC storage and answers the question of how much redundant data is stored in HPC storage systems, which could be found and removed by data deduplication techniques.

We focus on online data in HPC storage, which is stored in parallel file systems. Parallel file systems are widely used in HPC, striping data across several storage systems as well as offering concurrent access to the stored data. Examples for such file systems are Lustre [5], IBM's GPFS [6], PVFS [7] (now Orange File System [8]), and ceph [9].

Online storage for HPC involves several hierarchies. For example, there are usually (at least) file system volumes for home and project directories as well as scratch space. There are also different requirements for each of these volumes: Scratch space is usually meant for temporary storage and might even be pruned regularly, while home and project directories must be kept safe. The performance of the home and project directories is not as important as for the scratch space.

The workflow of a typical HPC application could look like the following:

1) Users store applications and input data in their home or project directories. Input data might need to be copied into an application-specific *run* directory.
2) Applications produce large amounts of intermediate data and store it in the scratch space.
3) Post-processing extracts relevant data to answer the scientific question. This reduces data that is then stored again in the project directories.
4) Input and output data may be moved (or copied) to a tape archive for backup and long-term archival purposes.

Storage installations in HPC centers usually range from several several TB to several PB.

To the best of our knowledge, **this paper presents the first large-scale study on data deduplication for HPC storage.** The deduplication potential is studied in multiple data centers of various sizes with different application domains. **Our findings about the redundancy in HPC systems could be a motivation to incorporate new deduplication architectures into HPC storage.** With an analyzed data set size of over 1212 TB of data, **this is the largest deduplication study we are aware of**.

II. Related Work

A. Deduplication

In contrast to other storage saving techniques, deduplication only stores unique data, while dropping data, which has been previously stored. The most common approach deduplicates data in two steps: First, the incoming data is split into small

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/ $31.00

parts called *chunks*, which, in the second step, are identified as unique or redundant based on their hash values.

Two general approaches can be used for this chunking step. The first one is called *static chunking* and splits the data into chunks of a fixed size. Although this technique is fast and simple, it is seldomly used because the insertion or deletion of a single character can already modify all following chunks. In this case, the deduplication system would classify all following chunks as new even if the following data is unchanged.

The second chunking approach is called *content-defined chunking* (CDC) and eliminates this problem. It was proposed by Muthitacharoen et al. for the Low Bandwidth File System (LBFS) [10]. With content-defined chunking, the system moves a sliding window over the data and hashes the windowed data in each step, using Rabin's fingerprinting method [11]. A new chunk boundary is found if the hash f fulfills the equation

$$f \bmod n = c \text{ for constant } c \text{ with } 0 \leq c \leq n.$$

This method generates chunks with an expected size of n bytes. Furthermore, it is common to enforce a minimal and maximal allowed chunk size. Content-defined chunking is used by most of the deduplication systems [1], [2], [12], [13].

The chunk size has a big impact on the deduplication ratio and the overhead associated with deduplication. Larger chunks reduce the amount of deduplicated data. On the other hand, large chunks reduce the overhead because fewer chunks have to be processed and fewer fingerprints have to be indexed. In enterprise backup storage systems, the average chunk size is typically set between 4 KB and 16 KB [1], [14], [15].

In the second deduplication step, each chunk is fingerprinted using a cryptographic hash function like SHA-1 or SHA-256 and then compared to already stored chunks by their fingerprints.

This approach, also called compare-by-hash, was discussed controversially [16]. It is possible that two different chunks produce the same hash value and that chunks can, therefore, be accidentally marked as redundant. The broad consensus is that the probability of such a silent data loss due to a hash collision is orders of magnitudes smaller than other sources of data loss, like undetected memory corruption [1], [17], [18], if a cryptographic fingerprinting function is used. The central observation for this estimation is the *birthday paradox*. It states that under the assumption of independently, uniformly identical distributed fingerprint values of length k bits, the probability that one or more hash collisions occur between the n data blocks is bounded by

$$p \leq \frac{n(n-1)}{2} \cdot \frac{1}{2^b}.$$

The probability of hash collisions when writing 1024 exabyte data to a deduplication system, which is based on 4 KB chunks and SHA-256 fingerprints, is smaller than 10^{-42}. Also important for deduplication is the 2^{nd}*-preimage-resistance*, which means that it is computationally infeasible for a fingerprinting function h to find for a given data block a a second data block $b \neq a$, so that $h(a) = h(b)$ [18], [19].

Some deduplication systems perform a byte-by-byte comparison to exclude any possibility of undetected hash collisions [20], [21]. If a full comparison is done, it is, e.g., possible to use shorter fingerprints, which reduces the overhead and problems of large fingerprint indexes [20].

The redundancy removal can be done at different points. The most common approach in backup systems is the approach to detect the redundancy directly while the data is written to the system (*inline deduplication*). This ensures that only the reduced data set is actually stored.

An alternative is *post-processing*, in which the data is directly written to the storage system as usual. A background process walks over the data, e.g., in idle times, and eliminates the redundancy eventually [22].

B. Deduplication studies

There are several studies that analyzed the deduplication potential in other contexts. The first study that covers more than a few GB was presented by Mandagere et al. [23]; they present deduplication results of a 450 GB personal computer storage for whole file duplicates, static and content-defined chunking.

In 2009, Meister and Brinkmann presented a deduplication study conducted on a file server of the University of Paderborn that was used by staff and students to store documents and other unstructured data [24]. The study focused on the temporal redundancy that is necessary for backup systems and provided an in-depth analysis of certain file formats to explain certain effects. Meyer and Bolosky present a study based on an engineering desktop workload at Microsoft [25]. They found that the removal of full file duplicates is highly effective for their workload. Wallace et al. recently presented a study focused on backup workloads based on statistics from over 10,000 EMC Data Domain installations [26]. Jayaram et al. analyzed the similarity between virtual machine images in an IaaS cloud environment [27].

All these data sets are inherently different from HPC workloads. Therefore, the results cannot be transferred directly.

C. Studies on other file related topics

The properties of file number and file sizes in HPC storage, which we also study in this work, have been analyzed by Ramakrishnan in 1992 [28] and by Dayal in 2008 [29]. Especially the latter is interesting because it provides information about the directory sizes, timestamp information, and sparse files that we did not investigate due to our different focus.

The IO characteristics of HPC applications have been evaluated in several studies [7], [30]–[33]. Outside the domain of HPC storage systems, several studies have been published on the distribution of file system metadata, file size, and file formats [34]–[36].

Kiswany et al. used deduplication techniques in their checkpoint storage system focused on desktop grids [37]. Their analysis is focused on the applications BMS [38] and BLAST [39] software and is limited to a small data set.

Another approach to reduce the storage requirements is data compression. In contrast to deduplication, compression reduces redundancy within a very narrow, local scope by increasing entropy. Compression can be used in conjunction with deduplication, e.g., compressing the chunk data after the deduplication detection.

Compression has been evaluated for HPC storage, especially for checkpoint data, in various publications [40], [41].

III. Methodology

The study is based on the open source tool suite FS-C [42], which has been developed by the authors. The tool analyzes existing data that is stored on the file systems of the HPC centers. It is not designed to actually deduplicate the data, but allows us to estimate the benefits of applying deduplication in an environment. FS-C processes the files of the HPC center and stores intermediate data on an additional storage for later analysis. This tool is a rewrite of the tools that have been created by the authors in 2008 for a similar but much smaller study on the data of the University of Paderborn and is in use since then.

A. Data Processing

The processing is done in two steps: a file system walk and an analysis step.

During the **file system walk**, all files are read, chunked, and fingerprinted. Multiple chunking approaches can be used optionally within a single file system walk. In this study, we focused on content-defined chunking with an expected average chunk size of 8 KB. To assess the impact of the chunking algorithm, some data sets have also been traced for static chunking and other chunk sizes.

Afterwards, all chunks are fingerprinted using SHA-1. To reduce storage requirements, only the first 10 bytes of the 160 bit SHA-1 fingerprint are stored (see Section III-C for a discussion of the potential impact of reducing the fingerprint size).

The output of the FS-C file system walk is a trace file containing all data that is necessary for later analysis:

- The hash of the file name, file length, file type, and a series of chunks for each file
- The fingerprint and the chunk length of each chunk

The file name is hashed to ensure privacy. The file type is determined by the file name extension by searching for a dot character within the last five characters of the file name. If no dot can be found, the file type is stored as "*Not available*".

A trace file is usually smaller than the original file set by a factor of 1000. For DKRZ's data, this means up to 7 TB, which may still pose a challenge in the following post-processing stage.

The data is analyzed in an additional post-processing step because building up a fingerprint index as an in-memory index would quickly exceed the available main memory. We have used the MapReduce paradigm for distributed data processing to analyze the fingerprints [43]. The data analysis script is written in the data flow language Pig Latin [44], which is then executed on a Hadoop cluster [45].

```
1  CHUNKS = LOAD 'RUN/chunks*' AS (filename, fp,
       chunksize);
2  FILES = LOAD 'RUN/files*' AS (filename, filelength,
       fileextension);
3
4  FILES_A = FOREACH FILES GENERATE filename,
       FileSizeCategory(filelength), fileextension;
5  A = JOIN FILES_A by filename, CHUNKS by filename;
6  B = GROUP A BY (fp, filelength, fileextension);
7
8  CL = FOREACH B GENERATE fp, fileextension, COUNT($1)
       as usage_count, MAX($1.chunksize);
9  S1 = GROUP CL BY fp;
10 S2= FOREACH S1 GENERATE group, SUM(usage_count), MAX
       (chunksize);
11 S3 = FOREACH S2 GENERATE group as fp, chunksize /
       usage_count as storedsize;
12
13 T1 = JOIN S3 BY fp, CL BY fp;
14 T2 = FOREACH T1 GENERATE fileextension, usage_count,
        chunksize as chunksize, storedsize;
15
16 T3 = FOREACH T2 GENERATE fileextension, usage_count
       * chunksize as totalsize,
17         usage_count * storedsize as storedsize;
18 U1 = GROUP T3 BY fileextension;
19 U2 = FOREACH U1 GENERATE group as fileextension, SUM
       (totalsize), SUM(storedsize);
20 U3 = FOREACH U2 GENERATE fileextension, totalsize,
       storedsize,
21         totalsize - storedsize as redundancy;
22 U4 = ORDER U3 BY totalsize DESC;
23 DUMP U4;
```

Listing 1. Simplified Pig Latin script for calculating the deduplication ratio per file type

Listing 1 shows a slightly simplified script that calculates the data deduplication ratio per file type.

In the first two lines, data is loaded from files containing chunking information and additional metadata about the fingerprinted files. The chunk data is joined with the metadata to add file type information to all entries (Alias B). Afterwards, we calculate the usage count for each fingerprint (Alias S2) and the number of persistent bytes each chunk reference is responsible for, which is the fraction of the chunk size and the number of references to that chunk (Alias S3). A way to calculate the contribution of each reference has also been used by Harnik et al. [46].

The second half of the script iterates a second time over all chunks and calculates the total size and the stored size grouped by the chunk fingerprint and the file extension. The total size and the stored size are accumulated per file extension. The difference between the total capacity and the stored capacity is the redundancy within a data set that could be achieved with a deduplication.

The overall deduplication ratio is defined as $1 - \frac{\text{stored capacity}}{\text{total capacity}}$, which is equal to $\frac{\text{redundant capacity}}{\text{total capacity}}$. A dedu-

plication ratio of 25% denotes that 25% of the data could be removed by the deduplication and only 75% of the original data size would be actually stored.

B. Data Sets

The deduplication analysis has been performed in various data centers to increase the likelihood that the results can be generalized. These centers have different sizes and focus on different application domains. If possible, the available data is split up into several data sets that focus on a phase in the HPC storage lifecycle.

RENCI is a collaborative institute that provides a *cyber infrastructure* including HPC computing to several universities in North Carolina. The file systems at RENCI are 140 TB in total. We have analyzed a subset of 11 TB.

The **DKRZ** is the German Climate Computing Center. It is dedicated to earth science with the goal to provide high performance computing platforms, sophisticated and high capacity data management, and superior service for premium climate science.

The DKRZ-B data sets mainly consist of third-party projects carried out on the DKRZ's high performance computer Blizzard. DKRZ-M are scientific project data similar to DKRZ-B but from a different suborganization. DKRZ-K consists of various data including test data, visualization, and guest account data. In total, the system contains around 1,000 user home directories and 330 project directories. The total work file system contains 3.25 PB of data with additional 0.5 PB on a scratch file system. We analyzed 1104 TB of this data.

The **RWTH** data set has been generated from the HPC systems at the University of Aachen. The major application domains are chemistry and mechanical engineering. We completley traced the Lustre file system used at that site. The data set has a size of 53.8 TB.

The **BSC** data sets come from the earth science department of the Barcelona Supercomputing Center. This department mainly computes weather and pollution forecasts. The data sets have been obtained from the storage system of the MareNostrum computer, which was ranked number 5 in the Top500 list when installed in 2005.

The different file systems build some kind of hierarchy (from the data life cycle point of view). The **BSC-PRO** (used by around 20 users, 9.1 TB of analyzed data) and the **BSC-SCRA** (used by only 3 users, 3.0 TB analyzed data) are the file systems where the data currently being used is placed. **BSC-BD** and **BSC-MOD** (both used by around 30 users, 11.2 TB, resp. 20.1 TB) are file systems where data is kept until the researcher decides that it will not be needed anytime soon. Then it is moved to the hierarchical storage management (HSM). Thus, these file systems have a moderate to low access rate. There is no policy implemented to move data from one level to another; it depends on the user methodology and hints set by the system administrator.

C. Biases, Sources of Error, and Limitation

The huge amount of data produced in HPC centers makes it impossible to generalize the evaluation results without taking possible biases into account. While we have traced more than one PB data, this is still only a part of the available storage in the data centers. Only the runs of RENCI and RWTH ran through the complete data set. Harnik et al. show that any naive sampling approach leads to bad estimations in the worst case [46].

Our sample of data centers over-represents European data centers; only a single North American center and no center from Asia have been analyzed. Additionally, the limited number of data centers might introduce a bias and limit the generalization of the presented results.

Due to limited access, we have not been able to check for what Dayan calls *negative and positive overhead* [29]. Positive overhead means that a file occupies more block storage than required by its file size. The reason could be that a small file may use a full block in the storage systems and that more blocks are allocated than currently used, e.g., by the Linux `fallocate()` function.

Negative overhead means that a file uses less capacity in blocks on the file system than indicated by the file size. According to [29], this may be caused by an optimization that stores the contents of small files in their corresponding inodes or (probably more important) by so-called sparse files – files that contain holes in the middle of the file because they are just partly written.

Symbolic links are detected and excluded from the trace. However, hard file system links are not detected. Hard linked files would be counted as full file duplicates resulting in an overestimation of the redundancy. Nevertheless, as we can see in Section V-D, in all cases but one, the percentage of file replication is below 5% and it would be very unlikely that a large percentage of this full-file replication comes from hard links.

Each file system walk has taken days or even weeks. Therefore, changes to the data set are only partially reflected. Deletions after the file has been chunked are not reflected. New files are only included if the directory scanner has not already processed the corresponding directory. It can also happen that a user moves already indexed files into a directory that will be indexed in the future, which results in an overestimation of duplicates. Again, this would be reflected in the full-file duplication rate, and as we have seen, this percentage is quite low in all but one case.

Due to the file system walk, it is impossible to relate deduplication and file modification patterns. Since every data set is only processed once, it is not possible to analyze the temporal redundancy of files.

We are not able to analyze the deduplication properties at sub-data set level, e.g., on directory level, because these information are not collected. Since file metadata like owner, groups, and timestamps are not collected, it is not possible to relate deduplication and file age.

Another source of potential errors is caused by the reduction of the stored fingerprints to 10 bytes. This increases the likelihood of accidental hash collisions between fingerprints. Nevertheless, the birthday paradoxon inequation gives us an

upper bound of 0.7% to have one collision in a PB data set. However, the bias even if such a collision occurs is still sufficiently low.

IV. File Statistics

In this section, we have a first look at the files on the traced file systems and give basic statistics for the distribution of file sizes and for file extensions.

A. File Size Statistics

An overview of the data sets and their size distribution is given in Table I. For each data set, it contains the number of files in the trace, the total capacity, and the mean file size. In addition, the table contains the 10%, 25%, 50%, 75%, and 90% percentiles for the file count and the capacity used. The 50% percentile aka the median for the file counts means that 50% of the files are smaller than the percentile. For the capacity, the same median denotes the file size for which 50% of the capacity is occupied by smaller files and 50% by larger files.

A pattern that we see in all data sets is that the file sizes are highly skewed. Most files are very small, but the minority of very large files occupies most of the storage capacity: 90% of the files occupy less than 10% of the storage space, in some cases even less than 1%.

In the BSC-BD data set, we observe that about a fourth of the files is smaller than 8 KB. Even this is a large fraction of the overall file count. The capacity of these files is not relevant. Another third of the files has a size between 4 and 8 MB, where also a third of the total capacities lies.

Files smaller than 4 KB are the most common file size category in the data sets BSC-PRO (62%), BSC-SCRA (46%), RWTH (22%), DKRZ-A (15%), DKRZ-B1, DKRZ-B3, and DKRZ-B5. Even in data sets where the very small files are the most common, the majority are small files. In most data sets, the median file size is less than 256 KB. Even here the median is always less than 8 MB.

On the other end, large files are also common, often hundreds of GB or even a TB in size. In the DKRZ-K data set, for example, 10% of the total capacity are occupied by files larger then 426 GB.

From now on, and in order to simplify the analysis, we will group files into bins, which we call file size categories: These are exponentially increasing ranges of file sizes.

B. File Extension Statistics

The usage distribution of file extensions in the data sets is interesting from the viewpoint of data deduplication. File extensions are an approximate for the file formats that are used to store files.

Figure 1 shows the distribution of file extensions according to the capacity of the files in relation to the total data set size. All file types that occupy less than 3% of the total capacity are aggregated into *Oth.*; "*N/A*" indicates files that have no file extension.

The extensions .nc and .ncf, describing the NetCDF file format [47], are aggregated into one file type because the file format is the same. A large number of files has a suffix that only consists of digits such as .000 or .029. These extensions are aggregated into a file type called Numeric or short "*000*". These extensions represent many different things, like the ID of the task writing it or the iteration of a simulation run.

In most data sets, the nc/ncf file extension is a very popular file format. This format is the most used one in all BSC data sets. Only files where we cannot detect a file type use more space. In all DKRZ data sets except B2 and M2, NetCDF files use most of the capacity. In the DKRZ-B3 data set, even more than 80% of the capacity and more than 70% of the files are NetCDF files. The nc file format is within the top 16 of the file types in all data sets except RWTH.

We have analyzed the file types of the full work and scratch storage space at DKRZ, while we have only analyzed the deduplication behavior of a subset of this 7 PB. We have compared the file type statistics of this sample with the statistics of the complete work and scratch file systems. The nc/ncf files, .grb, and .tar are also the most important file extensions in the complete file set. This indicates that our data sets are a good sample of the DKRZ file systems.

The archive file format tar is the most prominent format in RENCI and DRKZ-B2. We have observed that tar files that are stored or retrieved are not deleted and reside together with the contained files.

The .grb extension indicates files that are in the GRIB file format [48], which is a prominent data format for meteorology.

The types are very different at the RWTH and RENCI. At the RWTH, the HDF5 file format (.h5) [49] occupies about half of the available space. RENCI is the only data set where compressed data plays a major role on the file system; about 25% of the data is stored in .gz files.

In most data sets a big fraction of the files that has no file extension. In four data sets, this unknown type constitutes the biggest fraction of all file types. For example, the RENCI data set consists of 21.8% of unknown types.

Similar to the file without a file extension are the files with a numeric extension. These files may come from different file formats, and we cannot inspect these files in more detail. However, files with only a numeric suffix are very common as shown in Figure 1. It is even the "file extension" that occupies most space in the DKRZ-B4 data set.

V. Deduplication Results

The overall deduplication ratios based on content-defined chunking with an average chunk size of 8 KB for all data sets are shown in Table II.

In most data sets, the ratio ranges between 13.9% and 29.5%, except one data set with 7%, one with 49%, and one with 74%. Given that this data comes from totally different data centers, which are focusing on different application areas, it is surprising that the results are within a 15% range for most data sets.

The DKRZ-B3 data set is an outlier with a very high deduplication rate. A consultation with the responsible scientists revealed that the project of the B3 data set has been recently

TABLE I
FILE SIZE STATISTICS FROM DIFFERENT HPC DATA SETS. PERCENTILS IN 10%, 25%, 50%, 75%, 90%.

Data set	File Count	Capacity	Mean FS.	File Size Percentiles					Cumulative File Size Percentiles				
BSC-BD	865,063	11.2 T	13.6 M	3.6 K	7.6 K	11.1 M	13.3 M	33.8 M	12.1 M	13.3 M	33.8 M	257.4 M	692.6 M
BSC-MOD	1,113,760	20.1 T	18.9 M	8.1 K	64.7 K	84.4 K	126.4 K	5.6 M	55.1 M	732.8 M	2.6 G	7.0 G	27.3 G
BSC-PRO	3,987,744	9.1 T	2.4 M	46.0	368.0	1015.0	49.0 K	292.3 K	11.1 M	206.0 M	3.2 G	29.6 G	69.5 G
BSC-SCRA	76,957	3.0 T	41.5 M	0.0	299.0	9.7 K	1.1 M	139.3 M	133.8 M	156.8 M	356.5 M	4.1 G	27.3 G
DKRZ-A	343,118	27.0 T	82.4 M	2.6 K	26.4 K	3.3 M	31.1 M	76.0 M	42.5 M	444.8 M	1.0 G	6.4 G	22.6 G
DKRZ-B1	1,790,450	114.5 T	67.1 M	2.8 K	53.2 K	4.0 M	24.3 M	95.4 M	53.5 M	221.2 M	1.5 G	4.4 G	24.2 G
DKRZ-B2	2,273,444	109.1 T	50.3 M	4.5 K	114.0 K	4.3 M	12.8 M	54.8 M	40.9 M	144.2 M	3.4 G	32.5 G	85.7 G
DKRZ-B3	3,424,270	126.5 T	38.7 M	480.0	1968.0	6.0 K	35.4 K	1.3 M	376.9 M	1.2 G	4.6 G	22.9 G	53.8 G
DKRZ-B4	21,175,615	68.3 T	3.4 M	304.0	118.8 K	256.1 K	256.1 K	256.1 K	80.9 M	755.1 M	4.6 G	23.3 G	117.1 G
DKRZ-B5	650,372	75.3 T	121.4 M	1274.0	4.5 K	20.8 K	1.3 M	74.4 M	330.3 M	1.2 G	5.2 G	17.4 G	23.6 G
DKRZ-B6	1,340,788	47.9 T	37.5 M	3.1 K	36.5 K	2.0 M	10.5 M	29.7 M	31.0 M	265.7 M	1.9 G	13.4 G	39.9 G
DKRZ-B7	485,209	65.2 T	140.9 M	7.2 K	112.4 K	1.1 M	18.5 M	165.1 M	279.6 M	749.1 M	2.2 G	6.4 G	17.6 G
DKRZ-B8	3,445,787	176.6 T	53.7 M	0.0	0.0	178.0	930.0 K	25.3 M	174.7 M	721.8 M	2.0 G	4.0 G	14.5 G
DKRZ-K	2,245,564	42.9 T	20.0 M	492.0	2.1 K	9.2 K	180.2 K	982.8 K	465.2 M	4.2 G	45.9 G	94.8 G	426.0 G
DKRZ-M1	2,947,453	134.5 T	47.9 M	1062.0	6.8 K	76.3 K	2.4 M	38.1 M	66.8 M	351.9 M	2.4 G	6.7 G	44.3 G
DKRZ-M2	1,715,341	116.8 T	71.4 M	2.8 K	8.8 K	600.6 K	5.3 M	68.3 M	125.5 M	554.0 M	1.9 G	4.1 G	30.4 G
RENCI	1,024,772	11.1 T	11.4 M	4.3 K	37.2 K	124.3 K	752.5 K	3.7 M	37.8 M	103.6 M	254.3 M	3.7 G	37.7 G
RWTH	1,804,411	53.8 T	31.3 M	1072.0	7.7 K	1.1 M	5.6 M	8.0 M	43.4 M	771.0 M	18.0 G	96.0 G	512.0 G

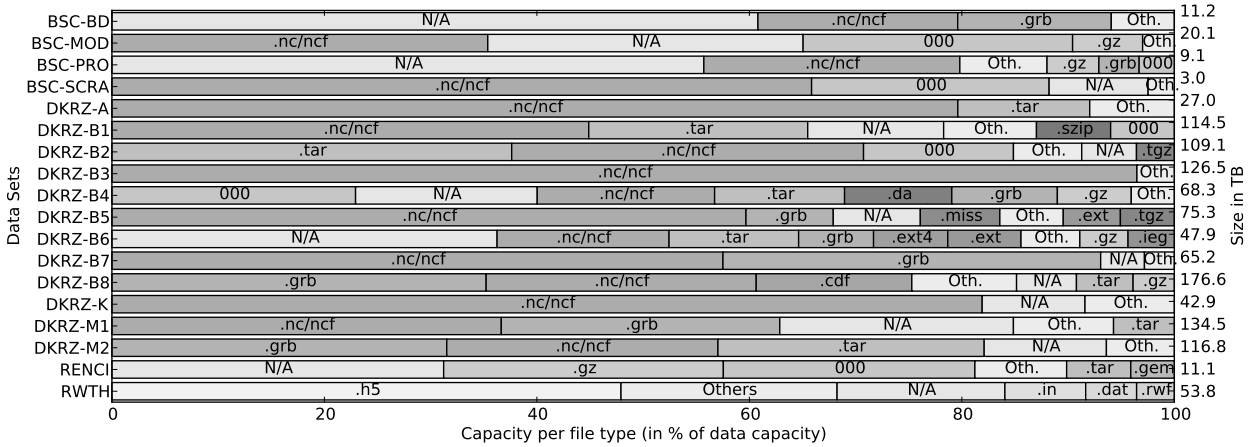


Fig. 1. File type distribution in all data sets.

TABLE II
DEDUPLICATION RATIO USING CONTENT-DEFINED CHUNKING WITH AN AVERAGE CHUNK SIZE OF 8 KB ON DIFFERENT HPC DATA SETS.

Data set	Ratio	Data set	Ratio
BSC-BD	7.0%	DKRZ-B5	29.5%
BSC-MOD	21.3%	DKRZ-B6	22.5%
BSC-PRO	29.3%	DKRZ-B7	14.1%
BSC-SCRA	38.3%	DKRZ-B8	13.9%
DKRZ-A	17.9%	DKRZ-K	49.3%
DKRZ-B1	19.7%	DKRZ-M1	15.0%
DKRZ-B2	27.6%	DKRZ-M2	21.1%
DKRZ-B3	74.4%	RENCI	23.8%
DKRZ-B4	27.1%	RWTH	23.2%

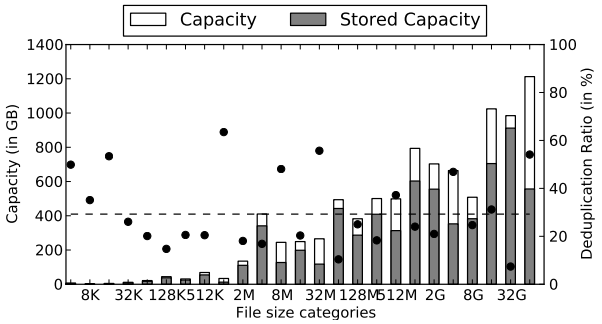
finished when the file system walk has been performed. Users have been in the process of rearranging data and deleting intermediate files.

Finding: In most data sets, between 15% and 30% of the data is stored redundantly and can be removed by deduplication techniques.

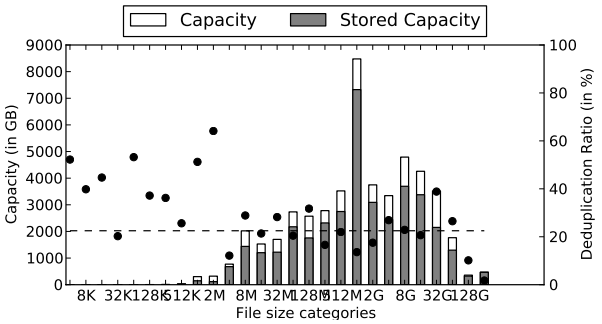
A. Deduplication by file sizes

In this section, the deduplication ratio depending on the file size is discussed. Due to the space limitation, it is not possible to present all findings and describe all observed patterns as well as notable outliers. While small files may place a burden on the metadata performance of HPC storage systems, they are not that significant from a capacity and deduplication standpoint.

Figure 2(a) shows the deduplication ratios of the BSC-PRO data set grouped by file sizes. Besides the logical capacity for files with a given size and the capacity that is actually needed to store the deduplicated data, additional information is included in the figure. The dot indicates the deduplication ratio within a file size category; for example, about 30% of the chunks from 16 GB files are redundant and could be deduplicated. The dotted line denotes the aggregated overall deduplication ratio over all stored data, which is 29.3%. It can be seen that the high deduplication ratio is caused by the large files. With ratios above 50%, the categories with 16 GB and

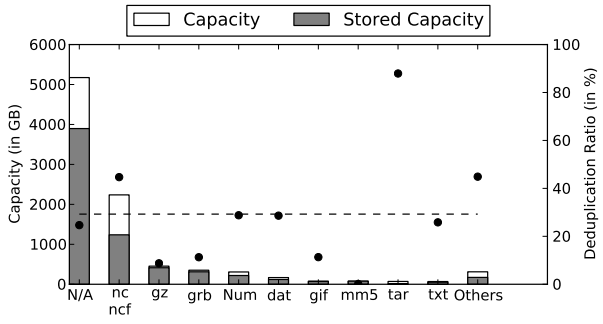


(a) BSC-PRO data set

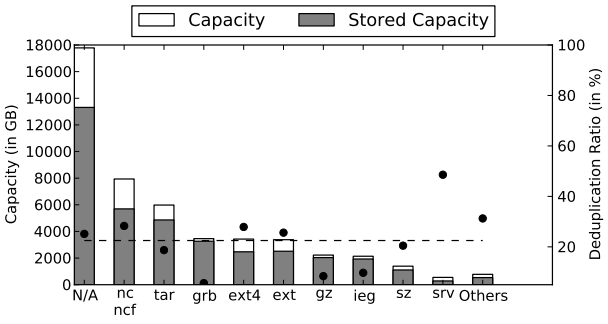


(b) DKRZ-B6 data set

Fig. 2. Deduplication results grouped by file size categories.



(a) BSC-PRO data set



(b) DKRZ-B6 data set

Fig. 3. Deduplication results grouped by file extension.

64 GB contribute most of the redundant storage.

In other data sets, it can be observed that large files contain a lot of redundancy, too. For example, in the DKRZ-A and the RWTH data sets, the ratios for the largest two categories are over 90%. The contrary is true for other data sets. Very little deduplication could be found for the largest files in the DRKZ-B4 and B6 (Figure 2(b)) as well as in BSC-BD, BSC-MOD, and BSC-SCRA.

On all data sets, small files (less than 1 MB) can be duplicated by a ratio of over 20%. In the DKRZ-B3 data set, small files can even be reduced by more than 80%. Such high ratios for small files indicate full file copies.

Finding: Often small files have high deduplication rates, but they contribute little to the overall savings. Middle-sized files usually have a high deduplication ratio. No clear picture emerges with regard to the set of largest files in the data set.

B. Deduplication by file extensions

In this section, we look at the results from the perspective of file extensions (file formats). Especially important in this section are not the most common file extensions, but file extensions that occupy the most capacity.

Figure 3 shows the deduplication results for the 10 most space consuming file types in the BSC-PRO and DKRZ-B6 data sets. The overall deduplication ratio is 21.3%, and most of the important file types actually have a comparable ratio. For example, 21.2% of the chunks in the nc files, which store data in the netcdf format, can be deduplicated. From the 20 most space consuming file formats in this data set, 13% have a deduplication ratio of 15% and higher. The most important exception in this data set are gz files, files that are compressed using the gzip algorithm, which uses a LZ77 variant [50]. GZip-compressed files only have a deduplication ratio of 5.7%. GRIB files only achieve a slightly better deduplication ratio. Depending on user settings, GRIB files could be compressed, which could explain the low deduplication ratio in this data set. As described in detail for other workloads [24], compressed files hurt the deduplication ratio. Small changes in nearly identically compressed data results in severe re-codings so that most data is not recognized as duplicate.

Table III shows the deduplication ratios for the most important file extensions: tar, GRIB, and NetCDF as well as for files where no file type could be detected.

The results show that (with the exception of BSC-BD) NetCDF files and files without file extension have a deduplication ratio that is usually higher than 20% over all data sets. This contrasts with, for example, the grb file type and tar archives where deduplication ratios range from almost nothing (grb 1.1%, tar 0.0% in BSC-BD) to over 87%. Tape archives require data transfers in larger granularity and data is usually aggregated using the tar format. A reason for the high deduplication ratio of (uncompressed) tar archives might be that the users also keep unpacked archives on the file system, leading to duplicated data.

Finding: nc files and files without a detectable file

TABLE III
DEDUPLICATION RATIOS OF FOUR USUALLY SPACE CONSUMING FILE TYPES IN DIFFERENT DATA SETS WITH CONTENT-DEFINED CHUNKING AND AN AVERAGE CHUNK SIZE OF 8 KB.

Data set	Deduplication Ratio			
	tar	grb	nc	N/A
BSC-BD	1.1 %	0.0 %	5.6 %	7.6 %
BSC-MOD	53.6 %	50.0 %	21.2 %	13.5 %
BSC-PRO	87.9 %	11.3 %	44.9 %	24.7 %
BSC-SCRA	80.8 %	87.8 %	51.2 %	22.6 %
DKRZ-A	18.2 %	1.7 %	16.4 %	28.7 %
DKRZ-B1	25.2 %	23.0 %	19.7 %	23.4 %
DKRZ-B2	25.6 %	27.2 %	38.2 %	21.6 %
DKRZ-B3	42.3 %	21.1 %	75.0 %	28.6 %
DKRZ-B4	14.1 %	24.1 %	37.3 %	25.3 %
DKRZ-B5	49.5 %	7.4 %	30.0 %	41.6 %
DKRZ-B6	18.7 %	5.6 %	28.3 %	25.1 %
DKRZ-B7	65.8 %	7.9 %	15.5 %	25.9 %
DKRZ-B8	21.3 %	5.6 %	20.7 %	26.8 %
DKRZ-K	57.2 %	9.2 %	47.7 %	77.4 %
DKRZ-M1	21.4 %	3.8 %	17.8 %	21.2 %
DKRZ-M2	25.0 %	6.5 %	26.5 %	45.0 %
RENCI	31.6 %	67.8 %	33.9 %	30.7 %
RWTH	45.4 %	n.a.	59.2 %	11.2 %

extension are the most space consuming file types in most data sets and the good deduplication results for these files boost the overall deduplication ratio. These are not the only files that contain redundancies.

C. Cross data set Sharing

Up to now, each data set was analyzed independently from the other. Now, the additional potential from deduplication due to chunk sharing between data sets is assessed. Cross data set shared chunks are chunks that occur in both data sets. A shared chunk would mean users from independent projects store identical regions in some files. This could happen, for example, when a scientific application is used in several projects and similar input data is used, e.g., boundary conditions for simulation.

If the deduplication domain is the individual data set, such a chunk will be stored twice. The redundancy removed within data sets is a lower bound of the possible global deduplication.

We analyzed the cross data set sharing for the two data sets DKRZ-B1 and DKRZ-B2. The individual deduplication rates are 19.7% and 27.6%; the aggregated rate is 23.6%. By removing all shared chunks, the deduplication ratio increases by 3.5 percent points to 27.1%.

The BSC data sets show a similar picture: If the domains of deduplication are the four data sets, the aggregated deduplication ratio is 20.5%. By combining all data sets into one large deduplication domain, the deduplication ratio increases to 22.4%.

Finding: In both data centers, chunks are shared across data sets, but the majority of the redundancy can be found within a local scope within projects.

D. Full File Duplicates

A simple form of deduplication is to eliminate files whose full content is a duplicate of another file. This is also called single instancing [51]. Based on the chunk fingerprints, we computed a fingerprint of the complete file data by concatenating all chunk fingerprints and then calculating a new fingerprint from the chunk fingerprint set. This is equivalent to fingerprinting the data itself.

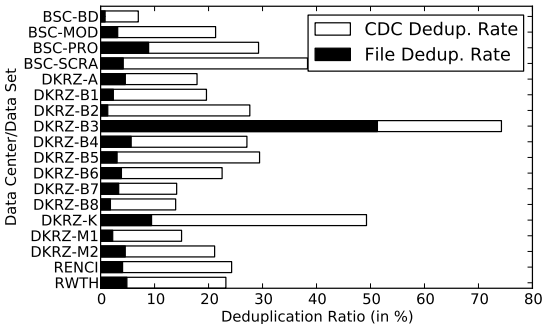


Fig. 4. Full file duplicates and CDC-8 deduplication ratio

Figure 4 compares the deduplication ratio using the content-defined chunking with the savings based on the elimination of full file duplicates. For example, for the RWTH data set 5% of data is duplicated, but with CDC an additional 17% of intra-file redundancy could be removed, resulting in a total reduction of 22%.

The results show that file duplicate elimination is not an effective technique in HPC storage because saving ratios drop by at least 80% on all data sets except the DKRZ-B3 data set. In the DKRZ-B3 data set, 51% redundancy could be found by file duplication elimination only (and 74% with CDC-8). This unusual behavior can be explained with file copies created by the scientists in preparation for archiving all the project data.

We also searched for full file duplicates shared between data sets for the BSC and the DKRZ-B data sets.

The analysis shows that cross data set file duplicates exist but not as many as expected. By combining all BSC data sets, an additional 1.8% deduplication rate is gained against 3.8% when the individual data sets are used. This is an increase by 50% but still small compared to the sub-file redundancy found with CDC-8.

The overlap between data sets can be as little as 0.02% when BSC-PRO and BSC-SCRA are combined. The largest overlap seems to be between BSC-MOD and BSC-PRO with 1.0%. We get similar results for the DKRZ data set: The full file duplicate ratio increases from 2.3% to 4.7%.

Finding: Full file duplication typically reduces the data capacity by 5% – 10%. In most data sets, most of the deduplication potential is lost if only full file elimination is used.

E. Other Chunking Strategies and Chunk Sizes

The chunk size and the chunking strategy used up until now is often used in backup systems because it presents a suitable tradeoff between deduplication rate and index overhead. Nevertheless, most of the redundant data comes from large continuous runs and a larger chunk size could find similar

deduplication with a much lower overhead. Therefore, we have used additional chunking strategies and chunk sizes to assess the behavior of the deduplication ratio depending on the chunk size for some of the smaller data sets.

While static chunking is not considered to be suitable for backup workloads, it performs well on virtual machine images (e.g., [52]). To investigate the impact of static chunking on HPC data sets, we have also investigated its impact.

It was infeasible to evaluate all combinations of chunk sizes and chunking strategies on all data sets. However, given that the deduplication rate is stable between data sets for the most part, this small sample also gives us an insight: The data set is a single 1.5 TB DKRZ project directory tree containing 10,000 files.

We evaluated this small data set with 4 different chunking strategy/chunk size pairs: CDC and static chunking with 32 KB and 64 KB.

The deduplication ratio is 42.2% and 40.8% for CDC-32 and CDC-64, respectively. As expected, the deduplication rate declines, but even with an average of 64 KB chunks a significant fraction of redundancy is detected. For fixed size chunks, the deduplication rate is significantly less than with content-defined chunking. Only 35.1%, respectively 33.8% of the redundant data blocks are detected with 32 KB and 64 KB fixed size chunks. Unpacked tar files are an example for which the rate declines with static chunking because file metadata is inserted into the data stream, which leads to different fingerprints for the fixed blocks.

The loss of detected redundancy is most severe in large files: For the largest file in the data set (80.0 GB), CDC-32 finds a redundancy of 19.6%, while no redundancy is found with fixed size chunks.

On the other hand, the disadvantage of fixed size chunking is smaller for the NetCDF files in the data set; the duplication is reduced from 31.2% to 23.6% using 32 KB chunks.

Finding: The deduplication slowly decreases slowly with increasing chunk sizes. Fixed size chunking detects around 6-8% less redundancies than content-defined chunking.

F. Zero-chunk elimination

It has been observed in enterprise backups and especially in virtual machine images that the chunk containing only zeros occupies a significant fraction of space in certain data sets [24], [52]. The zero chunk has the interesting property of always having the maximal chunk size if content-defined chunking using Rabin's fingerprinting method is used.

To check this hypothesis for HPC storage, we searched in the DKRZ data sets for the zero chunks.

The intuition is that, e.g., input files may contain matrix data. If these matrices are sparse and a lot of entries contain zeros, this also leads to zero chunks. In the DKRZ-A data set, 3.1% of the space is used by chunks containing only zeros. This also means that 3.1% of space can be reduced by simply eliminating blocks containing zeros. In the DKRZ-B3 data set, the zero chunk savings make up 24.3%. For the other data sets, the values range between 4.8% (DKRZ-B1) and 9.4% (DKRZ-B5). The data indicates that between a fifth to a third of the possible deduplication ratio comes from zeros alone.

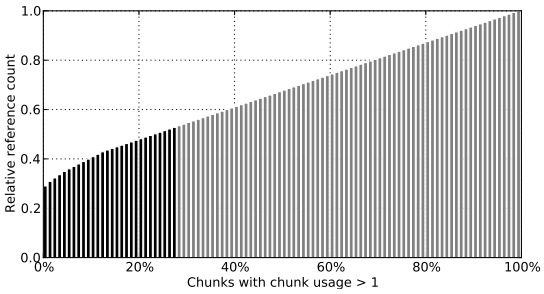


Fig. 5. CDF of the most referenced chunks in the DKRZ-B1 data set that are referenced at least twice. The light grey region only consists of chunks that are referenced twice.

It has been observed by Dayan that sparse files are used in HPC storage systems [29]. His data indicates that at most 0.5% of the data are sparse blocks.

Due to these sparse files, we cannot answer the question if these zeros really occupy capacity. If files are sparse, some fraction or even all of these zero chunks are not stored on disk but handled as sparse blocks by the file system.

Not all zero data is handled as sparse blocks by file systems. Most file systems only detect this if the user explicitly writes beyond the end of the file. Write operations where the data contains zeros are usually not detected.

This could mean that the file systems contain a large number of zero chunks beyond the zeros actually handled as sparse blocks. However, we cannot answer the question of sparse files for our data.

Finding: Between 3.1% and 9.4% (24.3% in DKRZ-B3 is counted as an outlier here) of the data are zeros.

G. Chunk Usage Statistics

We have seen that the zero-chunk contributes more to the redundancy in the data sets than other chunks and, thus, holds the most saving potential. Besides the zero-chunk, there might be chunks that occur more often than other chunks such that chunk usage distribution is skewed. To investigate this further, the chunk usage statistics of the DKRZ-B1 data set are assessed in depth.

Of all chunks, 90% were only referenced once. This means that the chunks are unique and do not contribute to deduplication. The most referenced chunk is the zero chunk, which has been discussed before. The mean number of references is 1.2, the median is 1 reference.

Figure 5 shows the CDF for the most referenced chunks that are referenced at least twice. As can be seen, the most referenced 5% of all chunks account for 35% and the first 24% account for 50% of all references. Of all multi-referenced chunks, about 72% were only referenced twice.

Finding: The chunk usage distribution is highly skewed. A small fraction of the chunks causes most of the deduplication.

VI. Discussion

We see that HPC data has a notable amount of redundancy. There are two viewpoints on the results.

The first one is that redundancies within an HPC storage system and especially within a project indicate bad practice (and workflows). At least some unnecessary redundancies are caused by exact file copies. Another kind of "bad user behavior" keeps unpacked tar files on the storage. This happens, for example, when copying tar files between the tape archival system and the online storage and not removing the tar file afterwards. Deduplication analysis can be used in HPC centers to detect suboptimal workflows to support users. The results show that such suboptimal behaviors exist, but they do not account for the majority of the redundancies observed.

The other viewpoint is that it might be worthwhile to incooperate ideas of data deduplication into HPC storage system. However, given the capacity needed for state of the art HPC storage systems and the high performance requirements, we are aware that deduplication can only be carefully applied in certain parts of a HPC storage infrastructure. Classic deduplication architectures used in backup systems are not applicable to HPC storage because of throughput and scale requirements as well as the specific IO patterns of HPC storage.

The purpose of this paper is not to answer how to apply deduplication in HPC storage, but to show that it would be worthwhile to research and develop such systems. A few hints to consider based on the study results are:

- *Post-processing:* Most backup deduplication systems are inline, which means that the redundancy is removed at the time the data is written, to avoid the high overprovisioning that would be necessary otherwise. We do not think that this property is important in HPC storage. Deduplication as an additional post-processing step usually reduces associated overhead and minimizes impact on write performance. However, this may change if the computing capacity continues growing while the performance penalty of storing data (network + storage) does not keep the pace. In the future, we may take inline deduplication (and compression) very seriously to reduce IO.
- *Large chunks:* The overhead of deduplication can be reduced by using large chunks, but there is a trade-off as larger chunk sizes detect less redundancy. We believe that the optimal setting for a deduplicating HPC system are chunks much larger than the 8 KB, which are used in today's enterprise deduplication systems.
- *Project-local deduplication:* Our analysis shows that the majority of the deduplication comes from a local scope. Cross project and cross data set sharing provides part of the overall deduplication, but it does not cause the majority of the redundancies found. This contrasts, e.g., with the experiences from deduplicating virtual machine images, where most of the duplicates are not found in a single virtual machine image but by sharing between many different virtual machines (see, e.g., Jin et al. [52]). This could be a helpful hint for future research and development, as smaller project-local indexes might be enough to find the majority of redundancies.
- *Volume specific:* Since an HPC center offers several independent volumes with different characteristics, it might only be possible to integrate deduplication into specific volumes. Scratch folders are likely to be subject to constant changes and data is usually just kept for a short time until it has been analyzed with post-processing tools. However, data in project directories (and home-directories) is stored for longer periods and benefits from deduplication by increasing available storage space.
- *Zero-chunk removal:* Sparse files can be seen as an arcane form of deduplication. However, today large runs of zeros are not removed and transformed into sparse blocks. Our findings indicate that this might be worthwhile for a file system to avoid writing empty blocks. Checking if a block only contains zeros is essentially free with respect to CPU and storage.

The biggest obstacle is the integration into parallel file systems. Another question that must be answered is if data deduplication might destroy the striped IO patterns, which are carefully designed by HPC applications and file systems. Disk systems need large sequential accesses to achieve maximum performance. The question here is how much distortion in the IO pattern is allowed before a notable performance degradation from the application is visible. With an inline deduplication approach, this question has to be answered for read and write operations. By using post-processing this only applies to reads.

VII. Conclusion

Our study shows that there is a huge potential for data deduplication in HPC storage systems that is not facilitated by today's HPC file systems. In most data sets, between 20 and 30% of the data is redundant and could be removed by deduplication techniques. It is in fact the most surprising finding that – with three exceptions – the deduplication ratio is very similar over all analyzed data sets.

However, today's deduplication approaches and deduplication systems are not suitable for the need of HPC storage. We hope that our findings motivate storage researchers and storage engineers to investigate if and how deduplication can be integrated into HPC storage systems.

VIII. Acknowledgments

The authors would like to thank all participating data centers as well as their technical staff. Without their support, this work would not have been possible. We also thank christmann informationtechnik+medien for their hardware support.

This work was partially supported by Spanish MEC under grant TIN2007–60625, by the German Federal Ministry of Economics and Technology (BMWi) under project grant SIMBA and by the EU commission for the SCALUS early researchers graduate school.

REFERENCES

[1] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the Data Domain deduplication file system," in *Proceedings of the* 6^{th} *USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[2] P. Shilane, M. Huang, G. Wallace, , and W. Hsu, "WAN optimized replication of backup datasets using stream-informed delta compression," in *Proceedings of the* 11^{st} *USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[3] K. Eshghi, M. Lillibridge, L. Wilcock, G. Belrose, and R. Hawkes, "Jumbo store: Providing efficient incremental upload and versioning for a utility rendering service," in *Proceedings of the* 5^{th} *USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[4] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in san cluster file systems," in *Proceedings of the 2009 USENIX Annual Technical Conference*, 2009.

[5] P. Braam, "Lustre: A scalable, high performance file system." Available at http://www.lustre.org/docs/whitepaper.pdf, 2002.

[6] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the2002 Conference on File and Storage Technologies (FAST)*, 2002.

[7] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for linux clusters," in *Proceedings of the* 4^{th} *Annual Linux Showcase and Conference*, 2000, pp. 317–327.

[8] [Online]. Available: http://www.orangefs.org/

[9] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *Proceedings of the* 7^{th} *Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[10] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," *SIGOPS Oper. Syst. Rev.*, vol. 35, 2001.

[11] M. O. Rabin, "Fingerprinting by random polynomials," TR-15-81, Center for Research in Computing Technology, Harvard University, Tech. Rep., 1981.

[12] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "HYDRAstor: A scalable secondary storage," in *Proceedings of the* 7^{th} *USENIX Conference on File and Storage Technologies (FAST)*, 2009.

[13] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Proceedings of the* 17^{th}*IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS)*, 2009.

[14] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in *Proceedings of the* 1^{st} *USENIX Conference on File and Storage Technologies (FAST)*, 2002.

[15] J. Kaiser, D. Meister, A. Brinkmann, and S. Effert, "Design of an exact data deduplication cluster," in *Proceedings of the IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2012.

[16] V. Henson, "An analysis of compare-by-hash," in *Proceedings of the* 9^{th} *Conference on Hot Topics in Operating Systems*, 2003.

[17] B. Hong and D. D. E. Long, "Duplicate data elimination in a san file system," in *Proceedings of the* 21^{th} *IEEE /* 12^{th} *NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, 2004.

[18] J. Black, "Compare-by-hash: a reasoned analysis," in *Proceedings of theUSENIX Annual Technical Conference*, 2006.

[19] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[20] C. Alvarez, "Netapp technical report: Netapp deduplication for fas and v-series deployment and implementation guide," 2009.

[21] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein, "The design of a similarity based deduplication system," in *Proceedings of the* 2^{nd} *Israeli Experimental Systems Conference*, 2009.

[22] T. Yang, H. Jiang, D. Feng, and Z. Niu, "DEBAR: A scalable high-performance de-duplication storage system for backup and archiving," in *Proceedings of the2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010.

[23] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying data deduplication," in *Proceedings of the Middleware Conference Companion*, 2008.

[24] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proceedings of the* 2^{nd}*Israeli Experimental Systems Conference (SYSTOR)*, 2009.

[25] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *Trans. Storage*, Feb. 2012.

[26] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proceedings of the* 10^{th} *USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[27] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei, "An empirical analysis of similarity in virtual machine images," in *Proceedings of the Middleware 2011 Industry Track Workshop*, 2011.

[28] K. K. Ramakrishnan, P. Biswas, and R. Karedla, "Analysis of file I/O traces in commercial computing environments," in *Proceedings of the 1992 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, 1992.

[29] S. Dayal, "Characterizing HEC storage systems at rest," Technical Report CMU-PDL-08-109, Carnegie Mellon University Parallel Data Lab, Tech. Rep., 2008.

[30] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "File-access characteristics of parallel scientific workloads," *IEEE Trans. Parallel Distrib. Syst.*, Oct. 1996.

[31] P. C. Roth, "Characterizing the i/o behavior of scientific applications on the cray xt," in *Supercomputing*, November 2007.

[32] D. Feng, Q. Zou, H. Jiang, and Y. Zhu, "A novel model for synthesizing parallel i/o workloads in scientific applications," in *IEEE International Conference on Cluster Computing*, September 2008, pp. 252–261.

[33] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, "Six degrees of scientific data: reading patterns for extreme scale science io," in *Proceedings of the20th international symposium on High performance distributed computing*, 2011.

[34] J. R. Douceur and W. J. Bolosky, "A large-scale study of file-system contents," in *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 1999.

[35] W. Vogels, "File system usage in Windows NT 4.0," in *Proceedings of the* 7^{th} *ACM Symposium on Operating Systems Principles*, 1999.

[36] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," *Trans. Storage*, vol. 3, no. 3, Oct. 2007.

[37] S. Kiswany, M. Ripeanu, S. S. Vazhkudai, and A. Gharaibeh, "stdchk: A checkpoint storage system for desktop grid computing," in *Proceedings of the* 28^{th} *International Conference on Distributed Computing Systems (ICDCS)*, 2008.

[38] P. K. Agarwal, "Role of protein dynamics in reaction rate enhancement by enzymes," *Journal of the American Chemical Society*, vol. 127, no. 43, pp. 15 248–15 256, 2005.

[39] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403 – 410, 1990.

[40] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, no. 1, 2007.

[41] D. Ibtesham, D. Arnold, K. Ferreira, and P. Bridges, "On the viability of checkpoint compression for extreme scale fault tolerance," in *Proceedings of the Euro-Par 2011 Parallel Processing Workshops*, 2011.

[42] D. Meister, "Fs-c, file system chunking tool suite, version 0.3.9," http://code.google.com/p/fs-c/, 2011.

[43] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of the* 6^{th} *Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[44] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008.

[45] Apache Software Foundation, "Hadoop," http://hadoop.apache.org/.

[46] D. Harnik, O. Margalit, D. Naor, D. Sotnikov, and G. Vernik, "Estimating deduplication ratios in large data sets," in *Proceedings of the* 28^{th} *IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2012.

[47] "The netcdf users guide," Unidata, User guide, March 2012.

[48] "Grib format edition 1," World Meteorological Organization, Tech. Rep., 1994.

[49] "HDF5 user's guide," The HDF Group, Tech. Rep., November 2011.

[50] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[51] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single instance storage in Windows 2000," in *Proceedings of the* 4^{th} *Conference on USENIX Windows Systems Symposium*, 2000.

[52] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proceedings of the* 2^{nd} *Israeli Experimental Systems Conference (SYSTOR)*, 2009.