Picky: Efficient and Reproducible Sharing of Large Datasets using Merkle-Trees

Daniel Hintze
FHDW University of Applied Sciences
Fürstenallee 3 - 5
33102 Paderborn, Germany
Email: daniel.hintze@fhdw.de

Andrew Rice
Computer Laboratory
University of Cambridge
15 JJ Thomson Ave., Cambridge, CB3 0FD, UK
Email: acr31@cam.ac.uk

***Abstract*—There is growing demand for researchers to share datasets in order to allow others to reproduce results or investigate new questions. The most common option is to simply deposit the data online in its entirety. However, this mechanism of distribution becomes impractical as the size of the dataset increases or if the dataset is frequently changing as new data is collected. In this paper we describe PICKY, a new Merkle tree based system for sharing large datasets which allows users to download selected portions and to receive incremental updates. We demonstrate the viability of our approach by quantifying its benefit when applied to a number of large datasets used in the networking and measurement community.**

I. Introduction

The design and performance of computer systems and networks is commonly evaluated based on public datasets of significant size. In general most scientific domains, e. g. genetics [1], neuroscience [2], plant science [3] and computer science [4] deal with increasing amounts of data from experiments and simulations today. Since research is largely publicly funded, it has been argued that researchers have an ethical duty to share scientific data in order to maximize the scientific contribution and facilitate a broad use of data [5]. Moreover, funding bodies increasingly require data sharing to be an integral part of research projects [6]. Sharing data not only reduces the costs of science [7] and facilitates further research, it is also crucial for validating approaches and repeating results in order to improve reproducibility [8]–[10].

Sharing and accessing datasets the size of several terabytes up to petabytes, however, requires significant computational resources and is thus challenging from a practical point of view, even if researchers are only interested in a small fraction of the data. Researchers lacking an adequate institutional infrastructure or relevant technical skills frequently find it hard to access such datasets [11]. Efficient sharing techniques therefore facilitate access to scientific data for less-well equipped or funded researchers, making the ability to conduct excellent research less of a privilege of economic wealth.

When facing the question of how to make a digital dataset of substantial size available to other researchers, there are four different approaches to consider: Datasets can be shared offline by physically transferring storage media [12], which is slow and expensive. Sharing datasets directly through HTTP or FTP downloads is more common today, hosting data on either an institutional website or in a data repository. Downloading huge datasets, however, is cumbersome and error prone. Peer-to-peer approaches like BitTorrent have been proposed as alternatives [13], [14], but inflict a loss of control on the dataset owner. Finally, cloud processing reverses the process by bringing code to the data and thus is advisable for very large datasets, but processing is more costly and complex compared to other options [15], [16]. In this paper we present the design and evaluation of PICKY, a simple yet powerful approach for repeatable and efficient sharing of large evolving scientific datasets. The contributions of this paper are as follows:

- We describe a novel data organisation model that facilitates repeatable, verifiable and efficient sharing of large datasets, featuring incremental updates and selective downloads.
- We show that our approach is beneficial by applying it to three large datasets from different domains and calculating the benefits for a selection of network measurement studies in the literature. We find that PICKY would have saved researchers between 26% and 93% of network and storage costs.

II. Related Work

Early examples of processing systems that allow certain queries in multi-dimensional datasets are the Active Data Repository [17] and DataCutter [18], a middleware infrastructure for processing datasets stored in archival storage systems across a wide-area network. Co-Sites [19] is an online resource management system to facilitate collaboration among geographically distributed research sites. Scibox [20] is a cloud-based infrastructure that features data reduction functions to subset a dataset or perform computations within the cloud rather than locally, but does not feature intra-file subsetting. Another example for cloud-based processing are cloud-based heterogeneous computing frameworks [21], designed specifically for multimedia mining applications, as well as Sector and Sphere [22], storage and compute clouds that allow user-defined functions within and across data centres. PreDatA [23] is a middleware for preparing and characterizing data whilst being produced on a peta-byte scale. Somewhat related to our logical data model is the Logical Information Systems as a File System [24], that enables dynamic information queries on

intra-file level. We distinguish PICKY from distributed version control systems (such as Git) by its ability to distribute slices (along many dimensions) of a dataset whilst maintaining useful functionality such as versioning and distributing updates.

III. Logical Data Model

Since scholars in many cases are only interested in particular aspects of a larger dataset, enabling them to select and download only certain subsets saves resources and enables access to datasets otherwise too large to process using commodity hardware.

To enable selectivity, contextual knowledge about the content and structure of the dataset is required. Real world scientific datasets mostly come in form of some file structure. While files contain the actual content of a dataset, directory structures and file names are commonly used to encode metadata describing the content, allowing for selectivity on file level. Files usually either constitute atomic binary data, for instance images, or can be considered collections of self-contained *entries*. Examples are network trace files containing sequences of independent network packets [25] and event logs consisting of events capturing, e. g. mobile device usage [4]. Selectivity at the entry level is desirable, if only a subset of entries is relevant for a particular research question, for instance only network traffic to a single UDP port [26] or certain usage events [27].

In order to enable selecting subsets of a dataset not only on file level but also on entry level in a generic way, a model for associating metadata with files and entries is necessary. We expect these metadata to come in form of key-value attributes, defined and provided by the dataset publisher based on which dataset consumers are then able select which subset, i. e. which files containing which entries, they wish to download.

IV. Physical Data Model

A. Indexing

To prepare a dataset for publication, a *repository* and an *index* file are created based on the original dataset by applying the following steps:

1) File Processing: Each file of the dataset is associated with a number of dataset-specific attributes in key-value form that allow users of the dataset to assess the content of the file. In addition, the file's relative path, filename and timestamp are recorded in order to restore these attributes on client side.

2) Entry Extraction: Subsequently, each file is split into one or multiple atomic *entries*, depending on the file's content. Entries can be of equal size, separated by a line break or defined by a binary format (e.g. network packets). Since our system therefore can not know how to define entries, the content of the file is streamed through a function implemented by the dataset owner to provide associated attributes each time one entry passes the data stream.

3) Chunk Persistence: Entries are grouped by their set of attributes into collection of entries called *chunks*. In addition to the entry data itself each chunk also contains the length of the data stored and a sequence id identifying the relative positioning of the entry within the source file. The content of a chunk is compressed and written to disk when either the entire file is processed or the size of the compressed entries exceeds a threshold C_{max}, in which case an additional chunk for subsequent entries with the same attribute set is created. The file format used to persist chunks starts with a protocol version to allow future changes, followed by a string naming the applied compression algorithm and the length of the uncompressed content. Finally, the compressed entries are written to the file, as depicted in Figure 1.

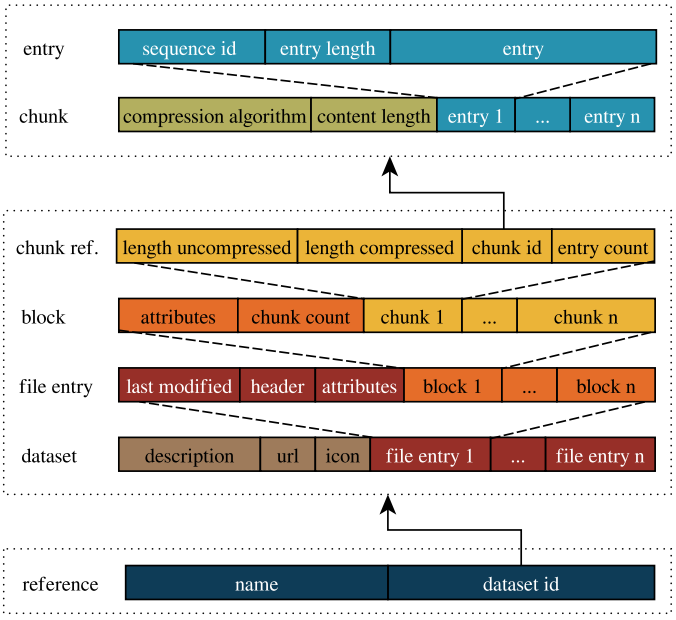


Fig. 1. Physical Data Model

4) Repository Storage: Chunks are stored in a *repository*, which is a form of content addressable storage system (CAS). A repository is basically a file structure, in which blobs, such as chunks, are stored using a content hash. This approach ensures deduplication to save storage and bandwidth while also allowing us to verify the integrity of the repository.

5) Dataset Index Structure: All chunks featuring the same set of attributes constitute a *block*. A block contains the associated attributes followed by a list of chunk hashes. If a client requests a particular attribute then the blocks containing that attribute are used to identify the chunks to send to the client. For each file within the dataset, a *file reference* containing metadata like name, relative path, timestamp, dataset specific attributes, binary file header and the blocks constituting the file is added to the index. All file references together form a *dataset index*, which also contains optional metadata describing the dataset, for instances a short description, an URL pointing to further reading and an icon image. The dataset index is compressed and stored as a single file in the repository, again using the hashed content as an identifier. The physical data model hence implements a Merkle tree.

To retrieve a dataset index file from the repository, its id, e. g. its hash, is required. To translate from human readable labels to repository ids, *references* are created. References are

stored alongside the repository and point to a dataset index file, being basically a substitution for symlinks, which are not available on every operating system. Figure 1 outlines the general picture of reference, dataset index and chunks.

B. Patches, Upgrades and Versioning

Existing sharing mechanisms offer limited to no support for applying patches and upgrades to published datasets in an efficient way, i. e. without requiring users to download the entire dataset again. There is usually no versioning scheme and so independently reproducing results is hard.

In PICKY, these properties are achieved by maintaining a single repository for different versions of the same or even multiple datasets. When a new version of a dataset is to be published, the indexing procedure is carried out again, reusing the existing repository. Unchanged chunks in the dataset produce identical content hashes as during previous indexing, and are stored only once due to the underlying CAS principle. Upgrades, i. e., appending new data to existing files or new files to the dataset, result in new chunks being stored in the repository. Patches, i. e., changing already published files, only affects the corresponding chunks and not entire files. For example, changing a single entry of an arbitrary large file at most requires clients to download one chunk of size C_{max} rather than the entire file. Entries are deleted by updating their length to zero.

Since the dataset index structure implements a Merkle tree, each distinct version of the dataset results in a new index file. Dataset owners would typically create references to each index file following some version naming scheme, e. g., using current date or incrementing a number. Since multiple references can point to the same index file, maintaining a head reference to the current version is possible. By using the hash to identify a particular index version, users are guaranteed to retrieve an exact copy of the specified version of a dataset, even if it has been altered meanwhile. Hence, this strong versioning scheme facilitates reproducibility of results for datasets under change.

C. Client Access

In order to allow clients to access the dataset, the repository containing the dataset index needs to be made available through an arbitrary file transfer protocol like HTTP. An obvious choice is using an off-the-shelf web server such as Nginx or Apache to deliver the repository as static content. However, filesystem, ftp and ssh access are also available.

A client-side application forms the counterpart of the server-side dataset indexing and enables the selective downloading and subsequent reconstruction of the dataset. Given a dataset reference, the client resolves the reference and obtains the dataset index file, which contains all information required to select subsets of the dataset down to file entry level.

D. Example

In this section we illustrate our data model by applying it to a simple sample dataset. It should be noted that in large real-world datasets, the number of files, blocks, chunks and entries is multiple orders of magnitude higher (see Table I). The sample dataset contains three small network trace files in pcap format, each containing a pcap header and a number of TCP and/or UDP packets, as outlined in Figure 2.

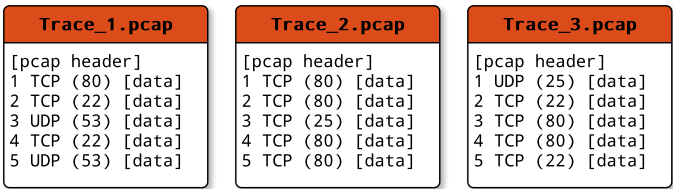


Fig. 2. Example dataset containing three network trace files

To create the repository and prepare the dataset for sharing, each file is parsed using an entry parser (usually provided by the dataset publisher). The entry parser yields the pcap header, file attributes (such as the last modified date) and most importantly the number of entries (in this case packets).

Each entry is associated with a number of attributes, namely the protocol family as well as the destination port. Entries are prefixed with their relative position within the source file. Entries are grouped by their attributes so that for each unique combination of attributes in the source file there exists at least one chunk file. When the size of a chunk file exceeds C_{max}, subsequent entries are written to a new chunk file with the same attribute set. In our example, we assume the attribute set `[TCP, Port 80]` in `Trace_2.pcap` exceeds C_{max}. As a result, the sample dataset is expanded into 9 chunk files as outlined in Figure 3, each stored using the hash of its contents as a reference.

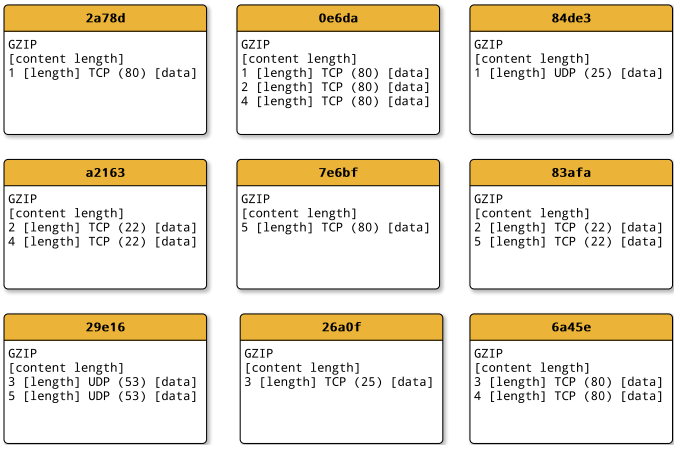


Fig. 3. Example dataset chunks

After parsing the entire dataset and creating the chunk files, the dataset structure is written to a single index file. The index file contains a logical file entry for each source file in the dataset and metadata such as the description of the dataset. Each logical file entry contains the pcap header of the respective source file, the filename, last modified timestamp and any other user-defined metadata. These are followed by a number of blocks, each representing a unique combination

of entry attributes. Blocks in turn are basically an ordered collection of chunk references (hashes of chunk files), as outlined in Figure 4. Once the index file is assembled, the indexing process is complete and the repository is ready for publishing.

A client intending to access the dataset first downloads the index file to learn the structure of the dataset. Based on this information, the client is capable of identifying which chunks are required to reassemble the dataset so that all entries featuring a chosen set of attributes are present. For the sake of this example, we assume the client is only interested in TCP traffic to port 22 (SSH). Using the index information, the client establishes that two blocks in two files contain the desired attributes. Knowing which blocks to restore, the client identifies all chunk references within these blocks and proceeds to download them. See Figure 5 for an outline of the relevant chunks.

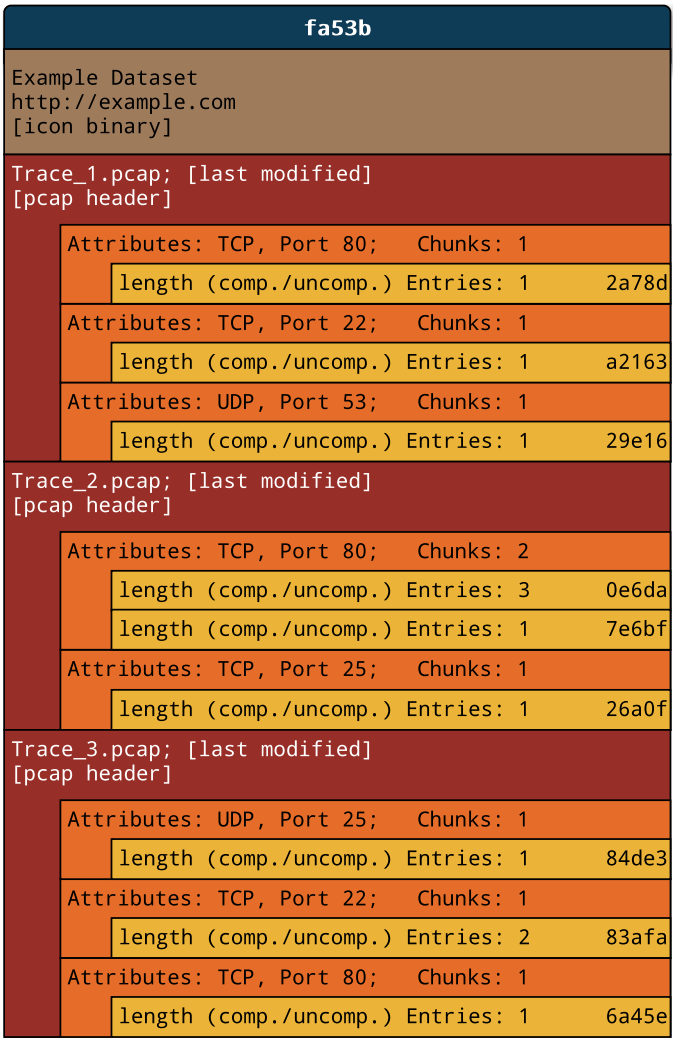


Fig. 4. Example dataset index file

Once all the required chunks have been downloaded, the subset of the original dataset is reconstructed, and valid pcap files are created. Each selected file is created using information

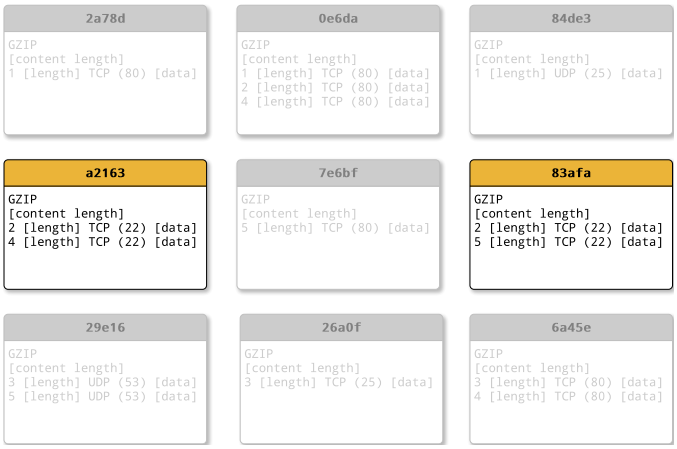


Fig. 5. Chunks containing TCP traffic to port 22

present in the file entry data structure, including the original binary file header. Subsequently, entries read from the chunks are sequentially written to the destination files, using the relative sequence id to preserve the original entry order when reading from several different block's chunks at once. Once all chunks have been processed, the dataset subset containing only TCP traffic to port 22 as outlined in Figure 6 is ready for further usage.

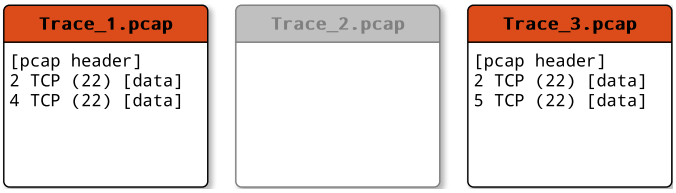


Fig. 6. Subset of the original dataset containing only TCP traffic to port 22

V. Implementation

We implemented this process in Java. The implementation consists of the following two parts:

A. Indexer

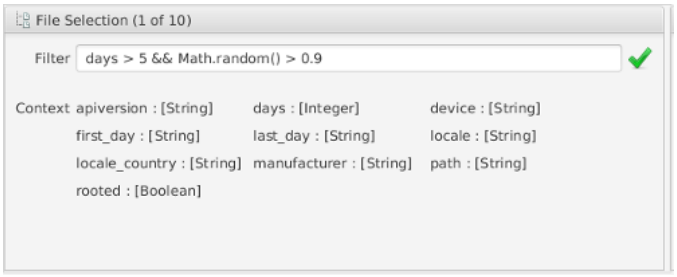
During index creation, a given dataset directory is traversed. Each file is run as a byte stream through a parsing interface, allowing dataset providers to apply their own definition of entries, to assign attributes for files and entries and a file header if needed. For common file formats like pcap network traces, however, we supply default implementations. The indexing process runs in parallel, being primary CPU-bound due to the amount of decompressing, hashing and compressing.

B. Client

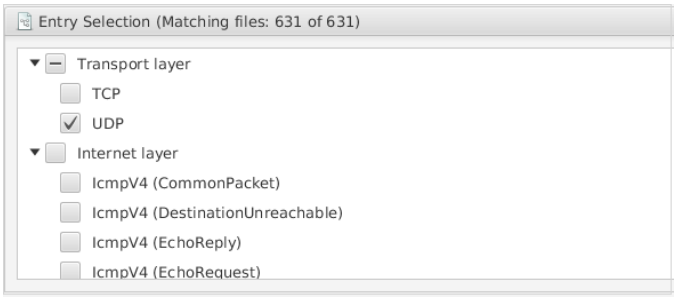
Since one of our goals was to simplify access to datasets of substantial size for research with a less technical background, we aimed to develop a download client that is robust, powerful and easy to use.

After downloading the single index file, the client generates a generic user interfaces that enables to define a subset of

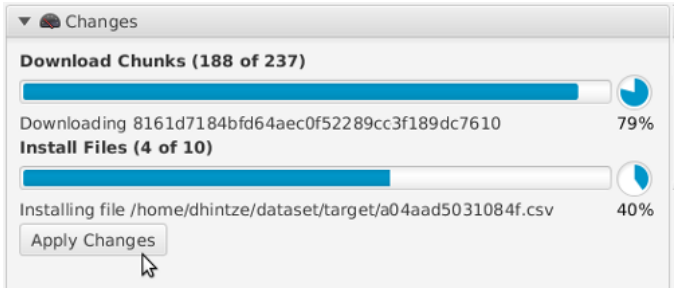
the original dataset based on the attributes on both file and entry level. Since number and diversity of attributes can be arbitrarily high, a powerful selection interfaces is desirable. Inspired by SQL WHERE clauses, we allow users to enter JavaScript statements as filter rules which are evaluated per file with its associated attributes being available in the executing context. This allows not only for sophisticated selection rules but also, for instance, drawing a random file subset using functions like *Math.random()*, as shown in Figure 7.



(a) File level subset selection



(b) Entry level subset selection



(c) Progress feedback

Fig. 7. Download client user interface

After specifying a subset of the dataset, the client compares the desired subset to the situation of the target directory and calculates any changes required in order to make the target directory match the subset. Possible changes are deleting files and directories not present in the subset, creating missing directories, installing new files and updating present files. Required chunks not already present in the local cache are queued for downloading.

Subsequently, the calculated changes are applied to the target directory. Most importantly, this includes downloading required chunks, verifying their integrity against the content hash, decompressing them and reassembling them to files. For this purpose, file headers are retrieved from the dataset index structure if present. Subsequently, the content of each file is reconstructed entry for entry based on selected blocks. Each block consists of one or more chunks, which are lazily read into memory while they are sequentially consumed. Sequence ids stored with each entry determine the order in which entries from different blocks are weaved together, ensuring correct reconstruction of the original file.

VI. Evaluation

Selectivity is a key feature of PICKY but this property is not found in current download-oriented dataset sharing approaches. We now demonstrate the value of selective downloading of larger datasets from the scientific community and the benefit in terms of bandwidth and storage savings. We do this by measuring the benefits of our proposed system for three real-world datasets of substantial size from different domains. For network and cluster traces, we analyse how previous studies in the literature would have benefited from selectivity. For mobile device usage logs, we analysed download logs from a custom download client featuring basic selectivity support, allowing us to derive accurate selection statistics.

A. Network Traces

The *CAIDA Anonymized Internet Traces 2014* contains anonymized passive traffic traces from two high-speed Internet backbone links recorded in pcap format, split into one-minute blocks, and compressed using gzip, totalling 631 files with a compressed size of 531.0 GB (1092.4 GB uncompressed). Besides, the dataset also contains a plain text file with statistical information as well as a file providing timestamps with nanosecond precision (since pcap requires microsecond precision) for each of the trace files. For simplicity, we only used the network traces but not the corresponding metadata and timestamps for the evaluation.

During index creation, traces were split into packets using libpcap. Each single packet was considered an entry and associated with available protocol information by assigning attributes denoting the respective network (e.g., IPv4, IPv6, ICMPv4) and transport layer protocol (e.g., TCP, UDP) as well as destination port number. We note that far more elaborate approaches would be possible here. All this would require is the implementation of an appropriate entry parser.

The segmentation resulted in 307 737 blocks with a total of 19.8×10^9 entries (packets). Index creation took 52 hours, mainly due to constantly crossing the boundary between code running on the JVM and native code by making calls to libpcap. Using a pure Java pcap parser would greatly increase indexing speed, but at the time of writing, no decent implementation was available to us. Applying a chunk size threshold $C_{max} = 5$ MB, the resulting index contained a total number of 402 215 chunks with a total size of 477.4 GB and an index description file of 23.7 MB. Interestingly enough, this represents a 9.98% decrease in size when comparing the compressed original dataset and the compressed index. Keeping in mind that we store an additional 221.4 GB (uncompressed) of meta data

by assigning a unique 64-bit sequence number and as well as a 32-bit integer holding the entry length to each of the 19.8×10^9 packets to allow for correct ordering of entries during reassembly. The reduction in storage space can be explained by gzip compression performing better on files with similar content (or, more precisely, a higher redundancy within 32K blocks). Since we are effectively grouping network packets by protocols, similarity increases, for instance through repeating header fields.

At the network protocol level, 99.66% of the stored data are IPv4 traffic while only 0.1% is IPv6 traffic. The fraction of IPv6 traffic is lower than what is reported in other studies, where IPv6 is found to be, for instance, 0.64% [28] of the traffic in 2014. However, our numbers are close to corresponding statistics by CAIDA[1]. Also, it was found that IPv6 adoption at the edges of the network is significantly less than in the network core [29]. 0.23% of the dataset consists of ICMPv4 traffic and only 0.006% of ICMPv6 packets.

On transport protocol level, we distinguished TCP and UDP traffic. We found 73.2% of the dataset contains TCP packets while UDP made up 11.4%. This leaves 15,4% of the dataset classified as *OTHER*. Again, this fraction could have been classified by putting more effort into parsing.

The third level of packet classification is based on TCP and UDP ports. While not guaranteeing correct results, ports are often used as an efficient way of deriving application protocol information from network traces—deep packet inspection not even possible unless application data has been collected [30]. For simplicity, we only considered well-known (0–1023) destination ports though more a fine grain classification would be possible. The most commonly occurring ports are 80 (HTTP) with a fraction of 15.6%, 443 (HTTPS) with 8.2% and 53 (DNS) with 0.5% of the dataset. Figure 8 outlines the resulting dataset fragmentation (not showing traffic to ports higher than 1023 as well as HTTP and HTTPS traffic).

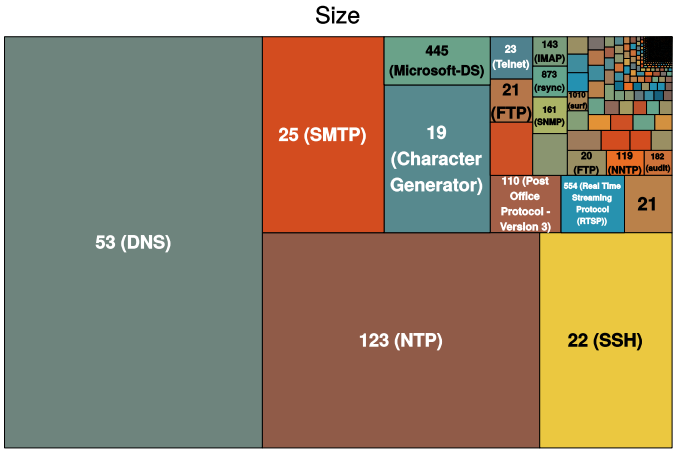


Fig. 8. Caida dataset fragmentation by destination port (excluding http/https and traffic to ports higher than 1023)

[1]https://www.caida.org/data/passive/trace_stats/

PICKY allows us to form subsets of the original dataset based on arbitrary filtering criteria from the three level of entry classification. For instance, a client interested in IPv6 traffic to UDP port 161 (SNMP) would only be required to download 34.5 kB instead of downloading the entire 531.0 GB network trace dataset. Even when including the index file, this still means a reduction in network traffic and client-side storage requirements by 99.996%. To determine realistic improvements for scientific applications, we analysed a number of published studies using large network trace datasets that based their research on only a part of the trace. For comparability, we assume the *CAIDA* dataset was used, even if the studies were based on an other version of the dataset or a different network trace dataset of significant size.

When it comes to downloading only subsets, a number of studies [31]–[33] can be found that analyse only traffic for a particular transport protocol, e. g. TCP. Using the corresponding subset would saved 182.5 GB or 34.37% of network traffic and client-side storage compared to the full dataset. Interestingly, some authors noted that they "did not have enough processing capacity to filter all CAIDA traces" [33], highlighting the point that computational resource limitations do affect the scientific community, motivating resource efficient sharing techniques. The potential benefits for researchers only interested in UDP traffic [34], [35] are even greater, since UDP traces account for only 10.24% of the dataset. Subsets based on application protocols derived from well-known port numbers are also applicable to real-world studies. One study analysed the CAIDA dataset for packets related to web and mail traffic [30], which is a subset 21.43% the size of the full dataset. We are, however, not able to construct a subset including peer-to-peer traffic comparable to what is used in [30], since the underlying protocols are not bound to well-known ports. In another the authors propose a method to generate realistic cover traffic for HTTPS, SMTP, and SSH [36]. By applying port-level selection, the relevant subset is only 7.5% the size of the original dataset.

B. Mobile Device Usage Logs

We also evaluated the proposed approach on the basis of the Device Analyzer Dataset.[2] Containing usage data from 30 393 Android devices collected over the course of 4 years by now, it is the largest publicly available dataset of this kind as of today [4]. Data are continuously collected and new versions of the dataset are published periodically, demanding both versioning and upgradability. For each participating device, the dataset contains a sequential log file of 263 key/value pairs along timestamps, recorded either periodically or event-based (see Figure 9). At the time of evaluation, the dataset features an uncompressed size of 11 531.7 GB and a compressed size of 1610.93 GB respectively. For index generation, a virtual machine running Ubuntu Linux with 8 cores (2.27 GHz) and 16 GB main memory was used. Dataset files were read from

[2]http://deviceanalyzer.cl.cam.ac.uk

and resulting repository files written to a NFS network storage mount.

For index creation, one entry was created for each of the $153.0\,\text{x}10^9$ log events within the dataset and assigned the associated key as an attribute. On file level, device metadata were assigned as attributes. On average, 46.8 blocks were created per device. Choosing a chunk size threshold $C_{max} = 5\,\text{MB}$ resulted in 1.5 million chunks with a total size of 2.4 TB being created, constituting an increase of 26.31% in size. This is in part due to less efficient compression of the smaller chunk files. However, meta information on entry level, namely sequence id and entry length also increase chunk size by additional 12 bytes per entry. Given the number of entries, these metadata add up to an uncompressed size of 1.7 TB (an overhead of 14%). Overall, indexing the entire dataset took around 45 hours.

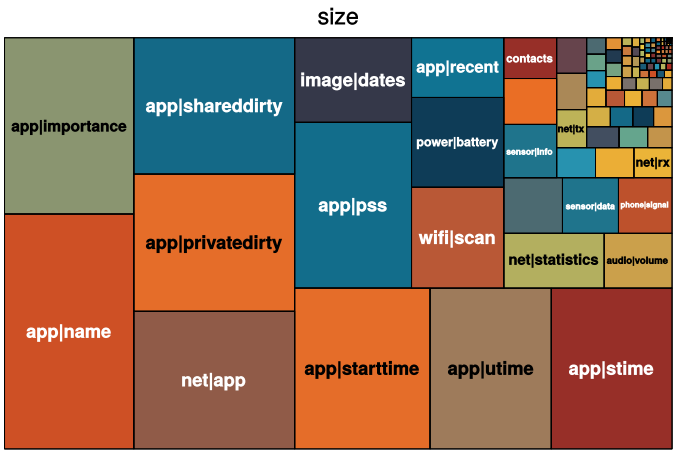
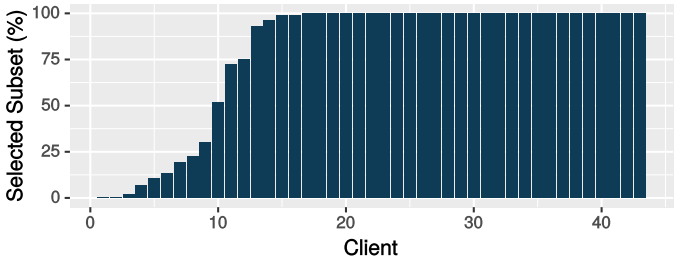


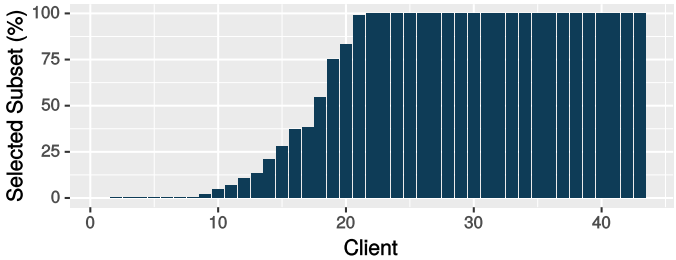
Fig. 9. Device Analyzer dataset fragmentation by event type

One rare example of selective dataset sharing is the custom download client used to make the Device Analyzer dataset available on occasion of the UbiComp/ISWC 2014 Programming Competition. The client recorded selection metrics, which we applied to our indexed version of the dataset in order to assess which proportion of the total dataset was requested for download. Of the 43 users accessing the dataset for the competition, 20 selected subsets of the dataset for download, making a total of 108 different selections. On file level (i. e device level), users limited the selected subset on average to 31.3% (median 25.37%) of the full dataset (see Figure 10a). Only taking entry level into account, selections limited the processed dataset to 39.4% on average with a median of 22.7% of the full dataset (see Figure 10b). Combining both file level and entry level selection results in an average selection of 23.65% of the dataset (see Figure 10c). This number is distorted by one client selecting almost the entire dataset, which is reflected by a median combined selection size of only 8.6%. Interestingly, 8 out of the 43 limited their selection to less then 1% of the original dataset size.

(a) File level selection



(b) Entry level selection



(c) File and entry selection

Fig. 10. Subset size under different selection level

C. Google Cluster Usage Traces

As a final example we show the performance of PICKY on a dataset of datacentre activity. Studying usage traces of real-world systems play an extensive role in understanding design challenges and evaluating novel algorithms and approaches. The first publicly available trace dataset from a multi-purpose cluster of significant size is the Google cluster trace dataset.[3] It contains anonymized traces for a month of activity in a single 12K machine cluster and provides information such as resource consumption, scheduling information, execution metrics, and constraints. Sensitive data like application or user names are obfuscated. The dataset consists of 2002 gzip compressed csv files with a total size of 44.1 GB (185.6 GB uncompressed), made available via Google Cloud Storage. Details about the content and semantics of the cluster traces can be found in [37].

For index creation, files were first grouped into 6 categories, distinguishing files containing job events, machine attributes, machine events, task constraints, task events, and task usage. Files were split line-wise into entries. The dataset contains only few discrete features suitable for entry-level tagging, hence tagging could only be applied to distinguish between task event

[3]https://github.com/google/cluster-data

	Original Format				Picky Format					
Dataset	Files	Format	Size	Compressed	Index	Blocks	Chunks	Entries	Compressed	Overhead
CAIDA 2014	631	libpcap	1092.4 GB	531.0 GB	23.7 MB	307 737	402 215	19.8×10^9	477.4 GB	-9.98%
Cluster Traces	2002	csv	185.6 GB	44.1 GB	1.4 MB	16 722	25 714	1.4×10^9	tab51.1 GB	15.87%
Device Analyzer	30 393	key/value	11 531.7 GB	1610.93 GB	118.8 MB	1.7×10^6	2.1×10^6	153.0×10^9	2034.8 GB	26.31%

TABLE I
COMPARING INDEX CHARACTERISTICS AND OVERHEAD FOR DIFFERENT DATASETS

type and job event type, as well as task execution constraint types. Processing was performed on a physical Ubuntu Linux server equipped with an Intel i7-3770 3.40GHz CPU and 16 GB main memory. A local SATA hard drive was used for persistent storage. With a chunk size threshold $C_{max} = 5$ MB, index creation took 5.5 hours, splitting 2002 csv files line wise into 1.4×10^9 entries, forming 16 722 blocks broken down to 25 714 chunks. The final index description file requires 1.4 MB of space while the entire index is 51.1 GB in size, resulting in an overhead of 15.87% compared to the original dataset format, again due to less efficient compression and 15.97 GB (8.6%) of additional metadata.

Since the dataset contains only few discrete features suitable for entry-level attribute association, the utility of entry-level subset selection is limited to applications in which only certain cluster event types are relevant such as the prediction of machine REMOVE events [38] (See Figure 11).

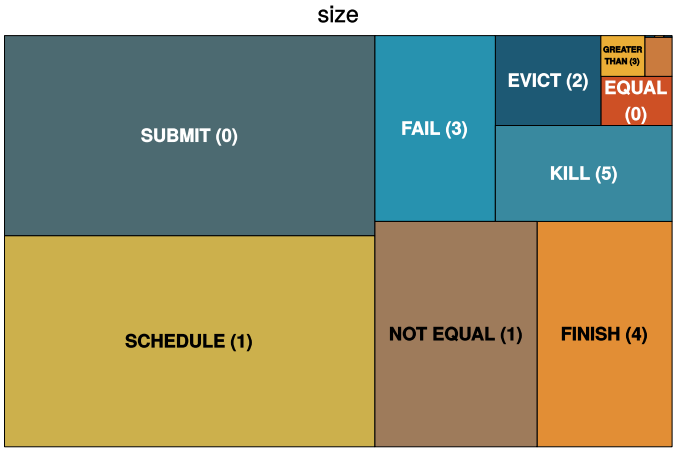


Fig. 11. Google Cluster Usage Traces fragmentation by event type

VII. CONCLUSION

In this paper, we presented PICKY, a novel approach for repeatable and efficient sharing of large evolving scientific datasets. PICKY features a number of properties that, depending on the nature of the dataset, are desirable. It allows dataset providers to publish updates without having clients to re-download content already present. Likewise, publishers are able to patch published data, for instance to correct errors. Consistency of the downloaded dataset can be verified to ensure correctness of received data. PICKY features versioning, which facilitates reproducibility of results by obtaining an exact copy despite any number of updates or patches. It also enables common access control schemes by working over arbitrary file exchange protocols. The most important feature, however, is the capability to download and process only a subset of the original dataset by enabling both file and intra-file selectivity.

We implemented PICKY and evaluated the concept by applying it to three different scientific datasets. We showed that our approach works well for datasets with an uncompressed size of 11 531.7 GB and handles 153.0×10^9 entries and more. Given a corresponding parsing function, PICKY handles text based files and arbitrary binary protocols equally well. Though our approach introduces 12 bytes of metadata per entry, we noted that for some datasets, we actually accomplish a reduction in size through the inherent compression-friendly file reorganisation.

We showed that there is value in selective downloading of scientific datasets by analysing both access statistics from the provider of a large dataset as well as previous studies analysing such datasets. We found that 40% of the users that specified a subset narrowed their choice down to less then 1% the size of the original dataset, resulting in significant reduction in traffic related cost, much faster access as well as lower computational resource requirements on client side. From looking at how network trace datasets are used in literature, we found that users who are working with a subset could save 26.8% to 92.5% of the size of the original dataset in both network traffic and local storage.

PICKY is released[4] under the Apache License, Version 2.0 and used successfully to provide the Device Analyzer dataset, now containing traces of more than 30 576 mobile devices, to interested researches.

ACKNOWLEDGMENT

The authors gratefully acknowledge funding by the German Federal Ministry of Education and Research.

[4]https://github.com/ucam-cl-dtg/picky

REFERENCES

[1] L. Clarke, X. Zheng-Bradley, R. Smith, E. Kulesha, C. Xiao, I. Toneva, B. Vaughan, D. Preuss, R. Leinonen, M. Shumway, S. Sherry, and P. Flicek, "The 1000 Genomes Project: data management and community access," *Nature Methods*, vol. 9, no. 5, pp. 459–462, 2012.

[2] A. R. Ferguson, J. L. Nielson, M. H. Cragin, A. E. Bandrowski, and M. E. Martone, "Big data from small data: data-sharing in the 'long tail' of neuroscience," *Nature Neuroscience*, vol. 17, no. 11, pp. 1442–1447, 2014.

[3] C. Ma, H. H. Zhang, and X. Wang, "Machine learning for Big Data analytics in plants." *Trends in plant science*, vol. 19, no. 12, pp. 798–808, 2014.

[4] D. T. Wagner, A. Rice, and A. R. Beresford, "Device Analyzer: Large-scale mobile data collection," in *Big Data Analytics workshop, ACM Sigmetrics 2013*, 2013.

[5] B. Brakewood and R. A. Poldrack, "The ethics of secondary data analysis: Considering the application of Belmont principles to the sharing of neuroimaging data," *Trends in plant science*, vol. 19, no. 12, pp. 671–676, 2013.

[6] National Institutes of Health, "Final NIH Statement on Sharing Research Data."

[7] R. Poldrack and K. Gorgolewski, "Making big data open: Data sharing in neuroimaging," *Nature Neuroscience*, vol. 17, no. 11, pp. 1510–1517, 1 2014.

[8] S. Fomel and J. F. Claerbout, "Reproducible Research," *Computing in Science & Engineering*, vol. 11, pp. 5–7, 2009.

[9] Yale Law School Roundtable on Data and Code Sharing, "Reproducible Research: Addressing the Need for Data and Code Sharing in Computational Science," *Computing in Science & Engineering*, vol. 12, pp. 8–12, 2010.

[10] R. J. LeVeque, I. M. Mitchell, and V. Stodden, "Reproducible Research for Scientific Computing: Tools and Strategies for Changing the Culture," *Computing in Science and Engineering*, pp. 13–17, 2012.

[11] M. V. Shapovalov, A. a. Canutescu, and R. L. Dunbrack, "BioDownloader: Bioinformatics downloads and updates in a few clicks," *Bioinformatics*, vol. 23, no. 11, pp. 1437–1439, 2007.

[12] M. R. Meiss, F. Menczer, S. Fortunato, A. Flammini, and A. Vespignani, "Ranking web sites with real user traffic," *Proceedings of the international conference on Web search and web data mining - WSDM '08*, p. 65, 2008.

[13] J. P. Cohen and H. Z. Lo, "Academic Torrents : A Community-Maintained Distributed Repository," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, 2014.

[14] M. G. I. Langille and J. A. Eisen, "Biotorrents: A file sharing service for scientific data," *PLoS ONE*, vol. 5, no. 4, pp. 1–5, 2010.

[15] A. P. Heath, M. Greenway, R. Powell, J. Spring, R. Suarez, D. Hanley, C. Bandlamudi, M. E. McNerney, K. P. White, and R. L. Grossman, "Bionimbus: a cloud for managing, analyzing and sharing large genomics datasets." *Journal of the American Medical Informatics Association*, pp. 1–7, 2014.

[16] R. L. Grossman, Y. Gu, J. Mambretti, M. Sabala, A. Szalay, and K. White, "An overview of the Open Science Data Cloud," *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, pp. 377–384, 2010.

[17] C. Chang, T. M. Kurc, A. Sussman, and J. H. Saltz, "Optimizing Retrieval and Processing of Multi-dimensional Scientific Datasets," *14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 405–463, 2000.

[18] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz, "DataCutter: Middleware for filtering very large scientific datasets on archival storage systems," *NASA conference publication*, vol. 9619020, pp. 119–134, 2000.

[19] Y. Zhang, M. Wolf, K. Schwan, S. Klasky, Q. Liu, and G. Eisenhauer, "Co-Sites: The Autonomous Distributed Dataflows in Collaborative Scientific Discovery," in *SC15 The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.

[20] J. Huang, X. Zhang, G. Eisenhauer, K. Schwan, M. Wolf, S. Ethier, and S. Klasky, "Scibox: Online sharing of scientific data via the cloud," *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*, pp. 145–154, 2014.

[21] H. Wang, B. Xiao, L. Wang, F. Zhu, Y.-G. Jiang, and J. Wu, "CHCF: A Cloud-Based Heterogeneous Computing Framework for Large-Scale Image Retrieval," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1900–1913, 2015.

[22] Y. Gu and R. L. Grossman, "Sector and Sphere: the design and implementation of a high-performance data cloud." *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 367, no. 1897, pp. 2429–2445, 2009.

[23] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "PreDatA - Preparatory data analytics on peta-scale machines," *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010*, 2010.

[24] Y. Padioleau, B. Sigonneau, and O. Ridoux, "Lisfs: A logical information system as a file system," pp. 803–806, 2006.

[25] "The CAIDA UCSD Anonymized Internet Traces 2014," *http://www.caida.org/data/passive/passive_2014_dataset.xml*.

[26] A. Herzberg and H. Shulman, "Vulnerable delegation of DNS resolution," *Lecture Notes in Computer Science*, vol. 8134 LNCS, pp. 219–236, 2013.

[27] D. Hintze, R. D. Findling, S. Scholz, and R. Mayrhofer, "Mobile device usage characteristics: The effect of context and form factor on locked and unlocked usage," in *Proceedings of the 12th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '14. New York, NY, USA: ACM, 2014, pp. 105–114.

[28] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey, "Measuring ipv6 adoption," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 87–98.

[29] A. Dhamdhere, M. Luckie, B. Huffaker, k. claffy, A. Elmokashfi, and E. Aben, "Measuring the deployment of ipv6: Topology, routing and performance," pp. 537–550, 2012.

[30] M. Dusi, F. Gringoli, and L. Salgarelli, "Quantifying the accuracy of the ground truth associated with Internet traffic traces," *Computer Networks*, vol. 55, no. 5, pp. 1158–1167, 2011.

[31] H. Jiang and C. Dovrolis, "Why is the internet traffic bursty in short time scales?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, p. 241, 2005.

[32] H. Ding and M. Rabinovich, "TCP Stretch Acknowledgements and Timestamps: Findings and Implications for Passive RTT Measurement," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 20–27, 2015.

[33] N. Ekiz and P. D. Amer, "Transport layer reneging," *Computer Communications*, vol. 52, pp. 82–88, 2014.

[34] C. Lee, D. K. Lee, and S. Moon, "Unmasking the growing udp traffic in a campus network," in *Proceedings of the 13th International Conference on Passive and Active Measurement*, ser. PAM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 1–10.

[35] M. Zhang, M. Dusi, W. John, and C. Chen, "Analysis of UDP Traffic Usage on Internet Backbone Links," *2009 Ninth Annual International Symposium on Applications and the Internet*, pp. 280–281, 2009.

[36] N. Schear and N. Borisov, "Preventing SSL Traffic Analysis with Realistic Cover Traffic," *16th ACM Conference on Computer and Communications Security*, 2009.

[37] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. a. Kozuch, "Heterogeneity and dynamicity of clouds at scale," *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC '12*, pp. 1–13, 2012.

[38] A. Sîrbu and O. Babaoglu, "Towards data-driven autonomics in data centers," in *IEEE International Conference on Cloud and Autonomic Computing*, 2015, pp. 45–56.