B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 413

DeLorean: A Storage Layer to Analyze Physical Data at Scale

Michael Kußmann[1] Maximilian Berens[1] Ulrich Eitschberger[2] Ayse Kilic[2]
Thomas Lindemann[1] Frank Meier[2] Ramon Niet[2] Margarete Schellenberg[2]
Holger Stevens[2] Julian Wishahi[2] Bernhard Spaan[2] Jens Teubner[1]

Abstract: Modern research in high energy physics depends on the ability to analyse massive volumes of data in short time. In this article, we report on *DeLorean*, which is a new system architecture for high-volume data processing in the domain of particle physics. *DeLorean* combines the simplicity and performance of relational database technology with the massive scalability of modern cloud execution platforms (Apache Drill for that matter). Experiments show a four-fold performance improvement over state-of-the-art solutions.

Keywords: data analysis, big data, LHCb, database, distributed computing, modern hardware, nuclear physics

1 Introduction

As part of the "Big Data" revolution, the way how "data" is being used in applications has seen a dramatic shift over the past years. Increasingly, relevant information is no longer carried in single pieces of data (*e.g.*, data points or records), but comes from the statistical relevance within very large data volumes.

Particle physics is a prime example of this trend that by today has reached virtually any application domain, from science to engineering to business. Analyses over massive sets of experimental data take the role that the close inspection of a single experiment had just a few years ago. To illustrate, the LHCb experiment at CERN's Large Hadron Collider (LHC) produces about 4 terabytes of raw data every second — year-round. Existing systems, even the largest ones, are still overwhelmed by data volumes of this scale.

Large data volumes obviously cry for database technology. Unfortunately, the required analyses are highly complex; scalable database technology lacks the expressiveness to support real-world analyses. As a joint effort with physicist from the LHCb experiment, we therefore developed techniques to bridge the large gap between complex analyses and query capabilities that can provide the necessary efficiency at scale.

In this work, we advocate the use of database technology to accelerate data processing in particle physics. And we report on *DeLorean*, our new, intelligent storage back end to accelerate data analyses at CERN.

[1] TU Dortmund University, Databases and Information Systems Group, {firstname.lastname}@cs.tu-dortmund.de
[2] TU Dortmund University, Experimental Physics V, {firstname.lastname}@tu-dortmund.de

- *DeLorean* builds on *Apache Drill* [Ap16a], the open source counterpart to Google's Dremel system [Me10]. Drill enables relational-style data processing at massive scale, leveraging technologies such as the *Hadoop Distributed File System (HDFS)* or *Apache ZooKeeper* for coordination.
- We show how real-world analysis tasks can be broken up into a *data-intensive part* — leveraging Drill's potential to *scan* massive data volumes in parallel — and into a *compute-intensive part* which covers most of the analysis's complexity but needs to run only on a fraction of the original data set.
- An important ingredient to *DeLorean* is an aggressive reduction of the data volumes that must be *scanned* during the analysis. We achieve this by applying *column-store technology* to a *synopsis* of the original data set, heavily optimized for scanning. In addition, we leverage *lightweight compression* to save bandwidth at the storage layer.
- We illustrate the potential of *DeLorean* using a *reference analysis*, on which we achieve performance improvements of up to a factor 4.6.

We will present *DeLorean* in the following order. Section 2 introduces into particle physics, a prime example for a new class of data-intensive applications. We show physical analyses can be made to scale by separating the data-intensive from the compute-intensive parts in Section 3. In Section 4, we discuss how scans can be optimized through column-oriented designs and compression. We evaluate our proposal on a prototype implementation in Section 5, discuss related work in Section 6, and wrap up in Section 7.

2 The LHCb Experiment at CERN

Particle physics, the *LHCb experiment* at CERN in particular, is an illustrative example of the data-centric nature of modern research in the natural sciences. The greater goal of the LHCb experiment is to understand the *matter-antimatter asymmetry*: why is there more matter in the Universe than antimatter?

2.1 Modern Particle Physics

To find an answer, physicists use the *Large Hadron Collider (LHC)* to accelerate and then collide *proton bunches*. During such collisions, new heavy particles are formed and *decay* shortly after into lighter particles. An example is shown here on the right (decay channel $B^0 \rightarrow J/\Psi\ K_s^0 \rightarrow \mu^+\mu^-K_s^0 \rightarrow \mu^+\mu^-\pi^+\pi^-$).

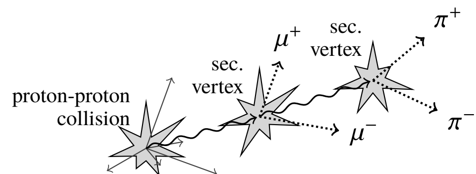

Fig. 1: Example of a collision experiment and its decay products. A B^0 meson is formed during a proton-proton collision, then decays into μ^+/μ^- and π^+/π^- particles.

Of particular interest to physicists are *rare* decay channels. And "rare" has to be taken very literally: probabilities of certain decays range between 10^{-12} and 10^{-15}. Therefore,

physicists produce a massive count of collisions, in the hope to eventually find (some) relevant events.

In practice, collisions are produced at CERN at a rate of up to 40 million collisions per second, year-round. Counting in down times of the accelerator, about 4×10^{14} collision experiments are performed per year, each of which weighs in about 100 kB of data. To handle the resulting massive data volume, data is processed in multiple stages as illustrated in Figure 2. A real-time *trigger system* filters the raw data stream right after data acquisition. Only a small percentage of all data survives this stage and is persisted to a large storage cluster.[3]

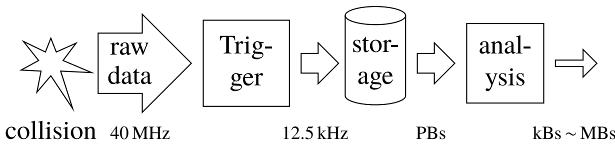

Fig. 2: Processing pipeline at CERN.

The daily task of a physicist is to run analyses on the stored data set. In practice, this means to write a (Python or C++) program that scans over the full data set to extract those collision events that are of interest to the particular question of the physicist. To illustrate, as few as ten $B^0 \rightarrow \mu^+\mu^-$ decays were extracted from the entire data set of the first LHC run from 2010 to 2013 [Th15].

2.2 Characteristics of Physical Analyses

Analysis programs typically filter the full, petabyte-scale data set according to complex criteria, strongly dependent on the particular physical question being asked. Thereby, (partial) criteria can be as simple as "return all events that produced a muon particle with an energy of at least ...," but also as complex as graph conditions on the 3d tracks that can be inferred — through compute-intensive algorithms — from the data points recorded. Figure 3 here on the right illustrates a strongly simplified example of an analysis that searches for $A^0 \rightarrow b^+b^-$ events. As can be seen in the pseudo code, simple predicates (*e.g.* on charge and mass) are interspersed with compute-intensive calculations.

```
for all evt in events do
    for all particle in evt.particles do
        if conditions on tracks and states then          ▹ first cut
            calculate information e.g. charge
            if charge < 0 then
                neg ← neg ∪ {particle}
            else if charge > 0 then
                pos ← pos ∪ {particle}
            end if
        end if
    end for
    for all pp in pos, for all np in neg do
        calculate combined mass np.mom. + pp.mom.
        if mass in A0 mass window then                   ▹ second cut
            emit (np, pp)
        end if
    end for
end for
```

Fig. 3: Simplified $A^0 \rightarrow b^+b^-$ analysis task (stripping line cut).

[3] Collision experiments filtered out during this stage are lost forever. About 20 PB–30 PB of data are stored to disk every year.

This type of complexity and diversity essentially rules out access structures like (multi-dimensional) indexes, leaving *scans* as the only viable search mechanism.

Analysis performance is, therefore, heavily influenced by the *volume* of the data that is being scanned. To reduce this volume, the existing platform at CERN uses a mechanism that physicists refer to as *stripping*: a *preprocessing stage* segregates all events into *stripping lines* on the storage cluster; each stripping line corresponds to pre-defined search criteria.[4] At this point, few hundred stripping lines are registered in the LHCb system, which was found to be a compromise between selectivity and the cost of pre-processing. In fact, stripping lines need to have a selectivity below 0.5 % to go easy on resources.

The stripping concept is both, a blessing and a curse. While it reduces the scan cost for common analysis (types), stripping *(a)* occupies scarce disk resources and *(b)* is limited to classes of analyses that have been pre-declared to the stripping process. In fact, most physicists would like to get rid of stripping rather sooner than later.

3 Making Analyses Scale

The software frameworks at CERN heavily rely on ROOT [RO16], which — for the context of this work — provides a persistence mechanism for C++ objects. That is, the existing storage layer at CERN consists of serialized C++ objects. For analysis (cf. Figure 2), these objects are de-serialized, then handed over to C++/Python code for processing. To save disk space, ROOT files are aggressively compressed.

The beauty of ROOT is its seamless interplay with the existing analysis code. Over time, a very large library of analysis routines has evolved that is mostly written in C++. It is, however, very difficult to make the approach run efficiently at very large scales, most importantly for two reasons:

(a) C++ object (de)serialization results in relatively complex data structures, on disk as well as in memory. The strategy can, therefore, poorly benefit from modern hardware advances and deep memory hierarchies. As a contrast, relational database engines intentionally stick to a very rigid and well-defined data model — one of the key ingredients to their excellent scalability.

(b) The ROOT de-serialization mechanism will read in C++ objects always as a whole. In practice, many kilobytes have to be read from storage, even when a very simple characteristic (*e.g.* a *charge* value) would be enough to decide that the object can be skipped for a particular analysis.

3.1 *DeLorean*: ROOT + Relational Storage

To avoid — or at least mitigate — the above two problems, *DeLorean* pairs the ROOT framework with a relational storage. On the relational side of *DeLorean*, we store a *synopsis*

[4] A stripping line compares best to a *materialized view* in a relational database engine.

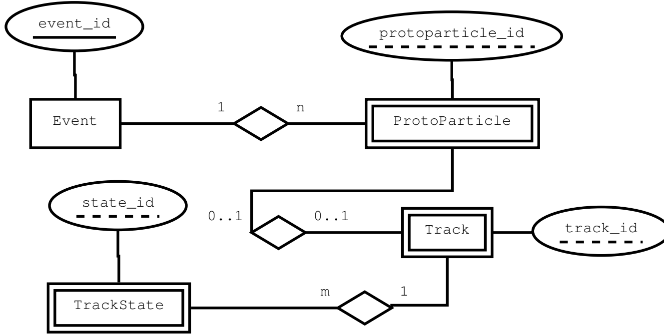

Fig. 4: Data model for the relational part of *DeLorean*.

of the full data set. The synopsis includes those fields of the data set which are queried frequently (using simple, "sargable" predicates) and with high selectivity. With a scan on the relational side, we select *candidate matches*, then de-serialize from the ROOT part only those C++ objects that are still promising.[5] If the relational scan is selective enough, this separation can result in significant savings in the overall scan volume.

Figure 4 visualizes the data model of the relational part of *DeLorean*. In the actual data set, for each collision event about 100 "proto particles" (particles whose kind might not yet be actually certain) and an equal number of particle tracks are being recorded. Over a year, about 10^{11} events are being persisted to the cluster storage. *ProtoParticle*, *Track*, and *TrackState* are factored out from *Event* as weak entities.

Queries over the data set are, again, essentially scans; the joins involved can be answered with efficient merge joins provided that the back-end knows about the physical storage order (by *event_id*). Scans of this type can be parallelized almost straightforwardly. Through (derived) horizontal partitioning, the data set can be distributed efficiently even over very large cloud installations.

In *DeLorean*, we use Apache Drill to realize the relational part. Based on a Hadoop Distributed File System (HDFS), Drill provides a natural way to parallelize typical analysis queries at very large scales. As we shall see in section 4, Drill's column-oriented *Parquet* storage format can assist in optimizing the type of scans typical for *DeLorean*. A big bonus of using a Hadoop-based approach is its fault tolerance which allows to employ cost-efficient consumer hardware.

At this stage of the project, we perform the separation of user analyses into data-intensive and a compute-intensive part (to be ran on the relational and the ROOT side of *DeLorean*, respectively) manually. *DaVinci*, the analysis software framework at CERN, however, already provides a domain-specific language component to express simple predicates over events. As one of our next project steps, we plan to use this language as a basis to automatically extract relational queries in *DeLorean*.

[5] ROOT supports efficient, tree-based seeking to selected events.

4 Optimizing Scans

The stripping line JPsi2MuMu in the original system is a good representative for a realistic event selection task. It matches the pattern shown in Figure 3 to select particle combinations where two muons are decay products of a J/Ψ meson. For the stripping line, the two cuts result in selectivities as listed in Table 1. Clearly, the two cuts reflect simple predicates with a very high selectivity (together around 1 : 3,000). As such, they are well-suited to pre-filter events using the relational store of *DeLorean*.

cut	**candidates**	**survivors**	**selectivity**
first	7,325,531	170,858	2.33 %
second	170,858	2,542	1.49 %
total	7,325,531	2,542	0.03 %

Tab. 1: Filter selectivity for the JPsi2MuMu stripping line cuts.

4.1 Column-Wise Storage

The characteristics in Table 1 make selection queries in *DeLorean* an excellent candidate to apply *column-store technology*. Storing data in a columnar fashion has two important advantages:

(a) Queries must read from disk only those columns that are actually relevant for the particular filter task (such as *charge* and the *position*-vector).
(b) When a query consists of multiple selection predicates (cuts), data for later attributes must only be fetched from disk for rows where earlier predicates were satisfied. The query optimizer may optimize the order of predicate evaluation accordingly.

Both properties result in a reduction of the data volume that has to be scanned (read from disk) for filtering. For simple queries like the ones we discuss here, a reduction of I/O volume may directly translate into an improved overall performance.

4.2 Lightweight Compression

Column-oriented storage goes well together with *lightweight compression*. With a reduced overall disk memory footprint, the system may be able to read the relevant data from disk with fewer I/O requests and faster. —Such an improvement will, of course, only be beneficial as long as the implied overhead — CPU cost for decompression in particular — does not outweigh the reduction in I/O cost. To this end, earlier work has developed compression schemes that are particularly lightweight and can provide high throughput. The most notable example is the PFOR family of compression schemes of Żukowski *et al.* [Żu06]. In *DeLorean*, we opted for Google's *Snappy* library [Gi16] and gzip, because they integrate particularly well with our base platform Apache Drill.

Traditionally, physicists at CERN have used LZMA for its significantly better compression rates. The high compression rates come at a significant CPU cost for decompression, making

LZMA an inferior choice from a runtime performance perspective. In Section 5, we will report on experimental results to support this claim.

5 Experiments

To test whether Apache Drill is a viable back end option for *DeLorean*, preliminary experiments were conducted to be able to compare DaVinci and *DeLorean*. In this experiment both approaches solely execute the "indexing part" (cf. Figure 3) to show the scalability of *DeLorean*. All experiments have been conducted on a Intel Xeon E5-2609v2 dual socket workstation (8 physical cores, no Hyper-threading) with 64 GB of RAM and a 7200 rpm SATA HDD running Scientific Linux 6.7.

The test data has been provided by the Experimental Physics V group at TU Dortmund University. To be able to run arbitrary indexing tasks, the whole data set is converted to Parquet. Parquet's columnar nature then allows Drill to only process needed columns and only work on a synopsis of the data. Cutting out the stripping process easily outweighs the two-fold data redundancy created by this proceed.

5.1 Compression Algorithms

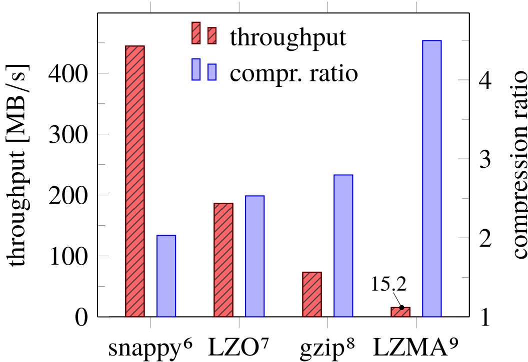

Fig. 5: Comparison of throughput vs. compression ratio for different algorithms on our data set.

Measurements using Intel VTune show that DaVinci spends 50 % of its runtime decompressing data using LZMA. Searching for a more CPU efficient compression algorithm, a small survey brought up three promising candidates: snappy, LZO and gzip. In an experiment we applied four different algorithms to uncompressed *Parquet* files containing actual LHCb data. Figure 5 compares the compression ratios for our data and shows the decompression throughput achieved on our test system (single threaded). LZO looks like a very promising candidate, compromising fairly good on decompression throughput and compression ratio. Unfortunately, Drill does not support LZO compressed Parquet files by now. Further experiments will stick to snappy and gzip for now.

[6] snappy-java 1.0.5.4 (`https://github.com/xerial/snappy-java`)
[7] lzop 1.03 (library version: 2.06) (`http://www.oberhumer.com/opensource/lzo/`)
[8] Oracle GZIPInputStream (`https://docs.oracle.com/javase/7/docs/api/java/util/zip/GZIPInputStream.html`)
[9] LZMA SDK 9.22 (`http://www.7-zip.de/sdk.html`)

Snappy exceeds the common HDD bandwidth by far, but has the lowest compression ratio in the test field and is therefore not a very good compromise for our test system. In section 5.2 we will see that *DeLorean* is in fact IO bound, therefore investigating algorithms with higher compression rates or specialized algorithms may be worthwhile.

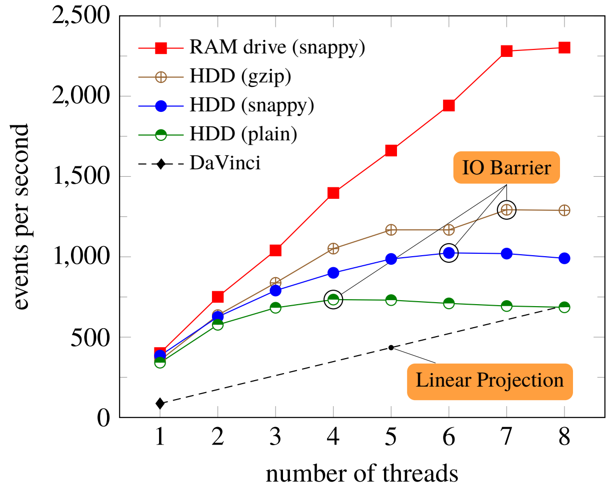

Fig. 6: Single node scalability of *DeLorean* compared to DaVinci.

5.2 Benchmarking *DeLorean*

Using an out-of-the-box embedded Drill instance we were able to achieve an event throughput increase of up to factor 4.6 (single-threaded). Unfortunately, a multi-threaded configuration of DaVinci is currently not available in our laboratory setting. For the sake of fairness we assume a linear scalability for DaVinci (best case). Figure 6 shows the scalability for different compression algorithms compared to the DaVinci projection. One notable result of the experiment is that *DeLorean* even outperforms the linear projection of DaVinci.

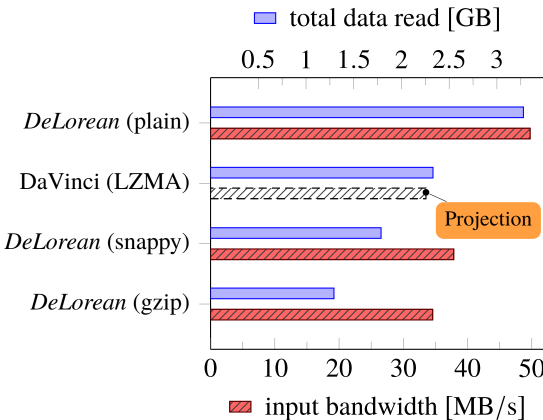

Fig. 7: Comparison of HDD traffic.

Figure 6 clearly shows that the HDD bound experiments hit the IO barrier.[10] Here we can see that compression can leverage the IO bottleneck significantly. Gzip, the “heaviest” of the compression algorithms, performs best in this scenario, so it might be worthwhile to invest in higher compression ratios using domain specific compression algorithms (like run-length encoding or delta encoding).

Talking about IO bottlenecks one should also look at the IO behaviour. Figure 7 shows the total amount of data read by the different approaches and the corresponding throughput at the IO barrier. Although DaVinci uses a very heavy weight compression algorithm, total reading IO is higher than for *DeLorean* using lighter weight algorithms. Due to the columnar storage design, *DeLorean* avoids reading irrelevant

[10] The RAM drive experiment is there to verify the IO bottleneck: Regarding the massive amount of data, it is unrealistic to process the whole data set in memory.

data from disk. Additionally, DaVinci spends about 50 % of its total runtime decompressing data that will never be touched later. Assuming the compression ratio of LZMA from Figure 5, DaVinci processes about three times more uncompressed data than *DeLorean*. Looking at the IO throughput in Figure 7, it becomes clear that the limiting factor cannot be the bandwidth but must be a bad access pattern.[11] We are currently working on strategies to enhance the access pattern by using intelligent caches and exploiting data inherent sortedness (our extracted tables are inherently sorted by ID). Additionally, we expect a lesser impact of access patterns when further scaling *DeLorean*, allowing for much bigger block sizes per thread.

6 Related Work

Notable related work has been done by Duggan *et al.* (*BigDAWG* [Du15]) in optimizing query execution over different storage systems with various data models. *DeLorean* is an example of what Duggan *et al.* call a *polystore system*: A polystore system uses multiple data sources with different data models (in our case relational Parquet files and object-oriented ROOT files) and takes into account all sources when optimizing queries.

There exist a number of competing Big Data platforms such as Apache Pig [Ap16b] and Impala [Ko15]. Both these platforms could serve as a back end for *DeLorean*. Notably, Apache Pig features LZO compression for Parquet files.

Nieke *et al.* [Ni15] analysed the CERN infrastructure with focus on the ATLAS project. They suggest using Apache Hive [Th09] in conjunction with Parquet to speed up data processing in CERN's datacentres.

The complexity of physical analyses rules out the use of common tree-based indexing mechanisms for acceleration, an observation that bears many similarities with the conclusions of Weber *et al.* [WSB98]. As a remedy for inefficient indices in high-dimensional spaces, they propose *VA-files*. As a compact data synopsis, VA-files allow for very fast scans. They are used to pre-filter data much like the column store of *DeLorean*.

Ekanayake *et al.* [EPF08] applied MapReduce-style processing (using Hadoop, among other implementations) to high energy physics data. Their experiments suggest good scalability for said analyses.

7 Summary

With our experiments we showed some weak spots of the current software setup at LHCb: Highly selective scans in conjunction with heavy weight compression. We showed that using (CPU-)balanced compression is essential for achieving high processing throughput.

Introducing *DeLorean*, a polystore system that uses modern database techniques in conjunction with columnar storage we were able to speed up data intensive processing steps in

[11] Common HDDs exceed 150 MB/s reading throughput for sequential reads.

DaVinci. *DeLorean* will bridge the gap between "flat" ROOT files and the relational database world, exploiting the advantages of both worlds, allowing for higher scan throughput and less data redundancy.

Acknowledgements: This work has been supported by the DFG, Collaborative Research Center SFB 876, project C5 (http://sfb876.tu-dortmund.de/).

References

[Ap16a] Apache Drill - Schema-free SQL for Hadoop, NoSQL and Cloud Storage, https://drill.apache.org, 15.09.16.

[Ap16b] Apache Pig, https://pig.apache.org, 20.09.16.

[Du15] Duggan, Jennie; Elmore, Aaron J; Stonebraker, Michael; Balazinska, Magda; Howe, Bill; Kepner, Jeremy; Madden, Sam; Maier, David; Mattson, Tim; Zdonik, Stan: The bigdawg polystore system. ACM Sigmod Record, 44(2):11–16, 2015.

[EPF08] Ekanayake, J.; Pallickara, S.; Fox, G.: MapReduce for Data Intensive Scientific Analyses. In: 2008 IEEE Fourth International Conference on eScience. pp. 277–284, December 2008.

[Gi16] GitHub - google/snappy: A fast compressor/decompressor, https://github.com/google/snappy/, 09.09.2016.

[Ko15] Kornacker, Marcel; Behm, Alexander; Bittorf, Victor; Bobrovytsky, Taras; Ching, Casey; Choi, Alan; Erickson, Justin; Grund, Martin; Hecht, Daniel; Jacobs, Matthew; Joshi, Ishaan; Kuff, Lenni; Kumar, Dileep; Leblang, Alex; Li, Nong; Pandis, Ippokratis; Robinson, Henry; Rorke, David; Rus, Silvius; Russell, John; Tsirogiannis, Dimitris; Wanderman-Milne, Skye; Yoder, Michael: Impala: A Modern, Open-Source SQL Engine for Hadoop. In: CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings. www.cidrdb.org, 2015.

[Me10] Melnik, Sergey; Gubarev, Andrey; Long, Jing Jing; Romer, Geoffrey; Shivakumar, Shiva; Tolton, Matt; Vassilakis, Theo: Dremel: Interactive Analysis of Web-scale Datasets. Proc. VLDB Endow., 3(1-2):330–339, September 2010.

[Ni15] Nieke, C; Lassnig, M; Menichetti, L; Motesnitsalis, E; Duellmann, D: Analysis of CERN computing infrastructure and monitoring data. In: Journal of Physics: Conference Series. volume 664, p. 052029, 2015.

[RO16] ROOT a Data analysis Framework, https://root.cern.ch, 19.09.16.

[Th09] Thusoo, Ashish; Sarma, Joydeep Sen; Jain, Namit; Shao, Zheng; Chakka, Prasad; Anthony, Suresh; Liu, Hao; Wyckoff, Pete; Murthy, Raghotham: Hive: A Warehousing Solution over a Map-reduce Framework. Proc. VLDB Endow., 2(2):1626–1629, August 2009.

[Th15] The CMS and LHCb collaborations: Observation of the Rare $B_s^0 \to \mu^+\mu^-$ Decay from the Combined Analysis of CMS and LHCb Data. Nature, 522:68–72, June 2015.

[WSB98] Weber, Roger; Schek, Hans-J; Blott, Stephen: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: 24th VLDB Conference. New York, USA, pp. 194–205, 1998.

[Żu06] Żukowski, Marcin; Héman, Sándor; Nes, Niels; Boncz, Peter A.: Super-Scalar RAM-CPU Cache Compression. In: 22nd International Conference on Data Engineering (ICDE'06). p. 59, April 2006.