Information Technology Department

A STUDY OF DATA REPRESENTATION IN HADOOP TO OPTIMIZE DATA STORAGE AND SEARCH PERFORMANCE FOR THE ATLAS EVENTINDEX

Z. Baranowski[1], L. Canali[1], R. Toebbicke[1], J. Hrivnac[2], D. Barberis[3]
on behalf of the ATLAS Collaboration

[1]CERN, Geneva, Switzerland; [2]LAL, Université Paris-Sud and CNRS/IN2P3, Orsay, France; [3]Università di Genova and INFN, Genova, Italy

ABOUT THE ATLAS EVENTINDEX

- The ATLAS EventIndex is a **catalogue** of all real and simulated **events** produced by the experiment at all processing stages.
- The system contains **tens of billions** of event records (6e^{10} records as of September 2016), each consisting of ~1000 bytes.
- The goal of the ATLAS EventIndex is to allow **fast** and **efficient selection** of events of interest, based on various criteria, and provide references that point to those events in millions of files scattered in a world-wide distributed computing system.

OBJECTIVE

- Data formats represent one important area for optimizing the performance and storage footprint of applications based on Hadoop.
- This work reports on the production usage and on tests using several popular data formats and storage solutions including Map Files, Apache Parquet, Apache Avro, Apache Kudu and various compression algorithms in order to improve the performance of storing and searching data within the ATLAS EventIndex system.

STORAGE EFFICIENCY AND DATA ACCESS PERFORMANCE MEASURED

The **same** datasets have been stored on the **same** Hadoop cluster using different storage techniques. The **data access** tests were performed with **Apache Impala**.

Hardware specification: cluster of 14 machines with 2 x 8 cores @2.60GHz, 64GB RAM, 2 x 24 SAS drives

Storage technology applied

Average encoded record length (bytes) — Lower is better

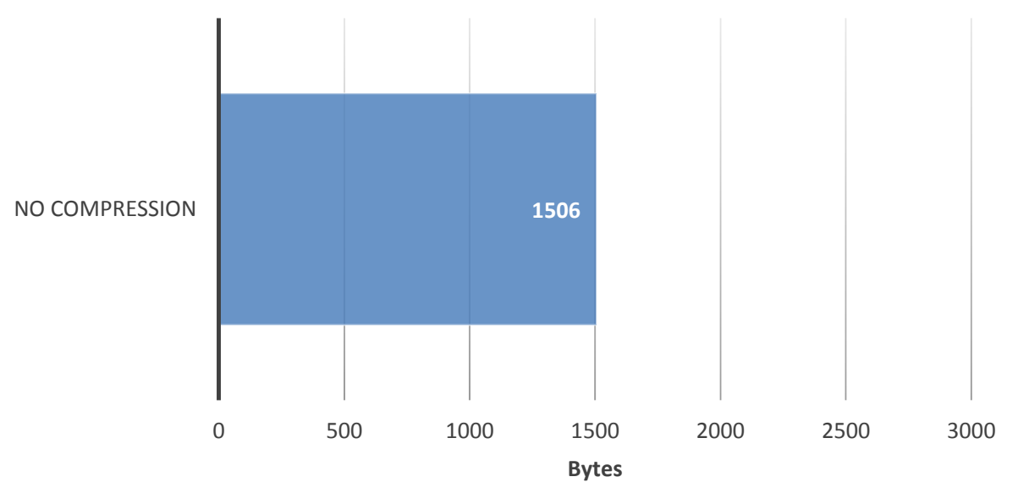
Average data writing speed per EventIndex dataset (records / s) — Higher is better

Average data lookup time (s) * — Lower is better

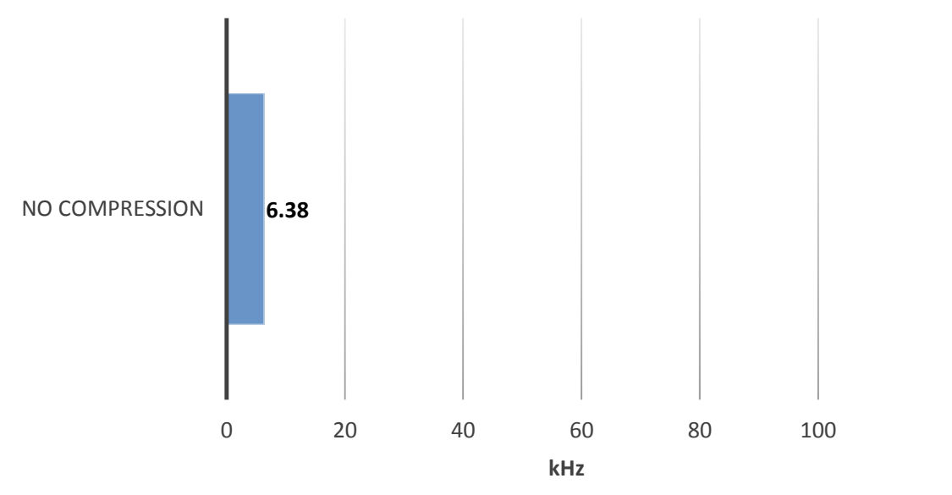
Average data scan speed per CPU core (records / s) ** — Higher is better

MapFile format on HDFS

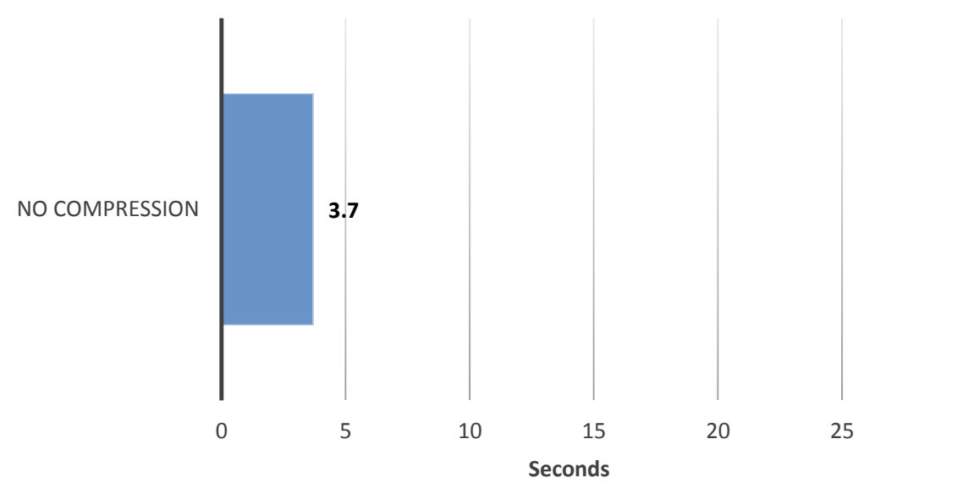
File format currently being used in the EventIndex production system. Stores records in flat set of key value pairs. Keys are indexed so the data of given record can be directly accessed by a key.



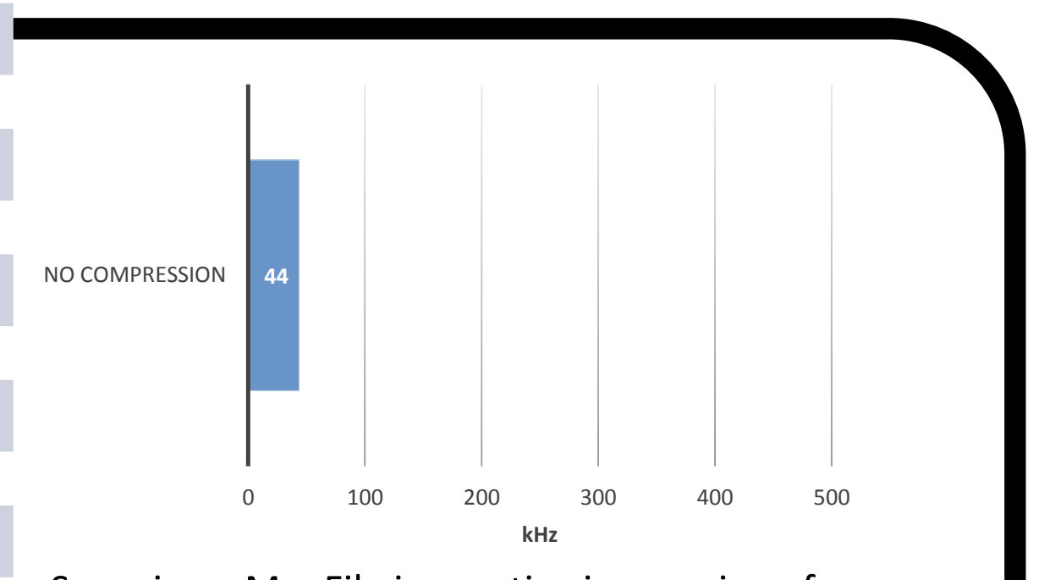
Average EI record occupies a lot of data due to the extensive information about HLT attached to each event.



Storing data in MapFiles requires sorting records by key beforehand. For big data collections it can imply a significant overhead and slow down an ingestion.



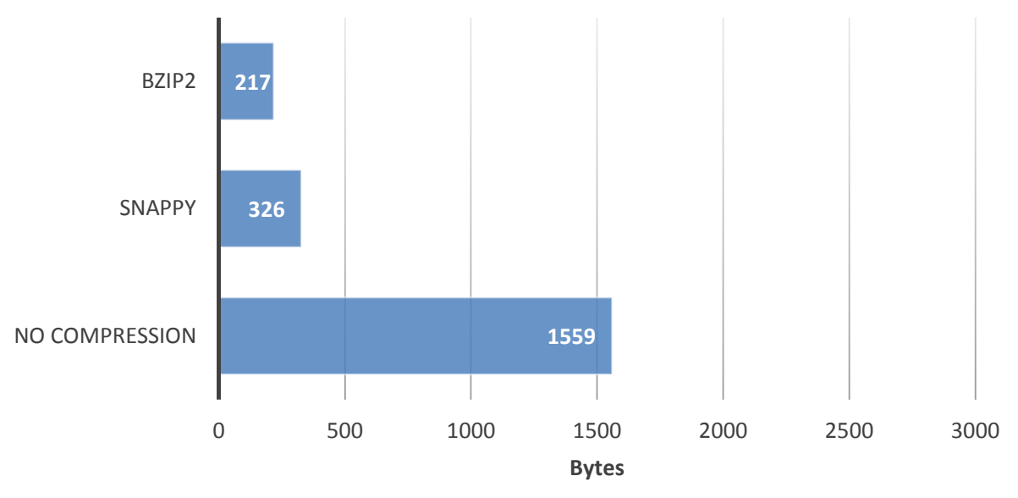
Data lookup performance depends on the size of index file that has to be downloaded and processed on the client side.



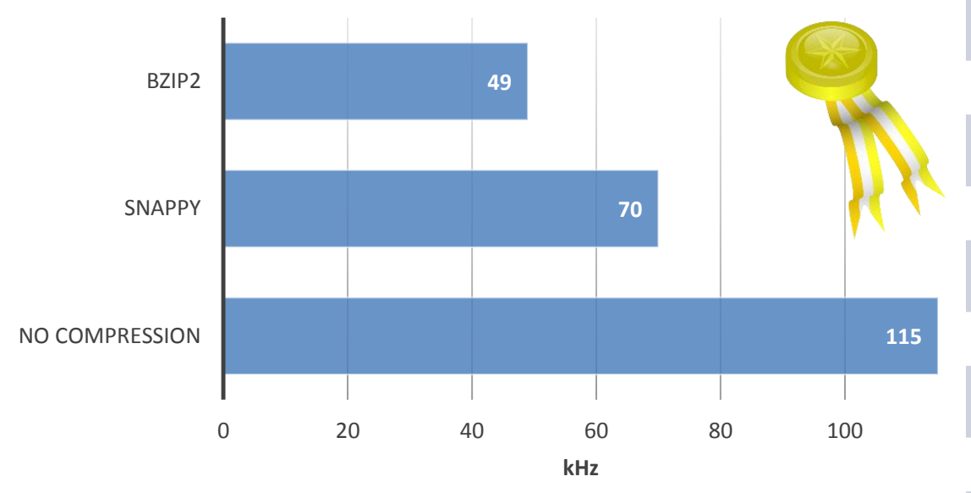
Scanning a MapFile in practice is scanning of underlying sequence file containing the data. Due its simple encoding it has similar performance like processing a text file.

Apache Avro file format on HDFS

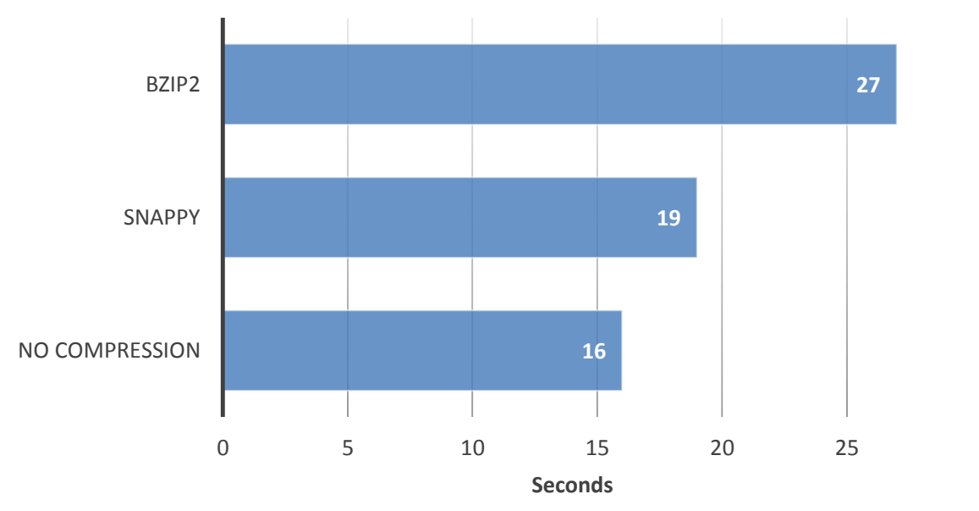
Data serialization standard for compact binary format widely used for storing persistent data on HDFS as well as for communication protocols.



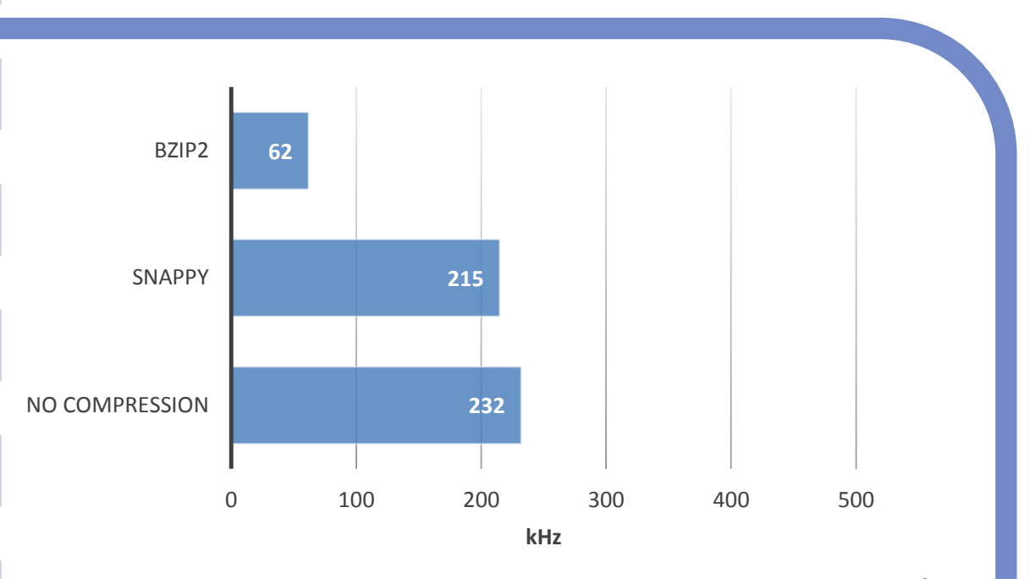
Avro encoding has similar level of efficiency as map files. Significant savings can be obtain by using compression (Snappy, BZIP2).



Encoding data in Avro format is relatively lightweight and gives the best ingestion performance



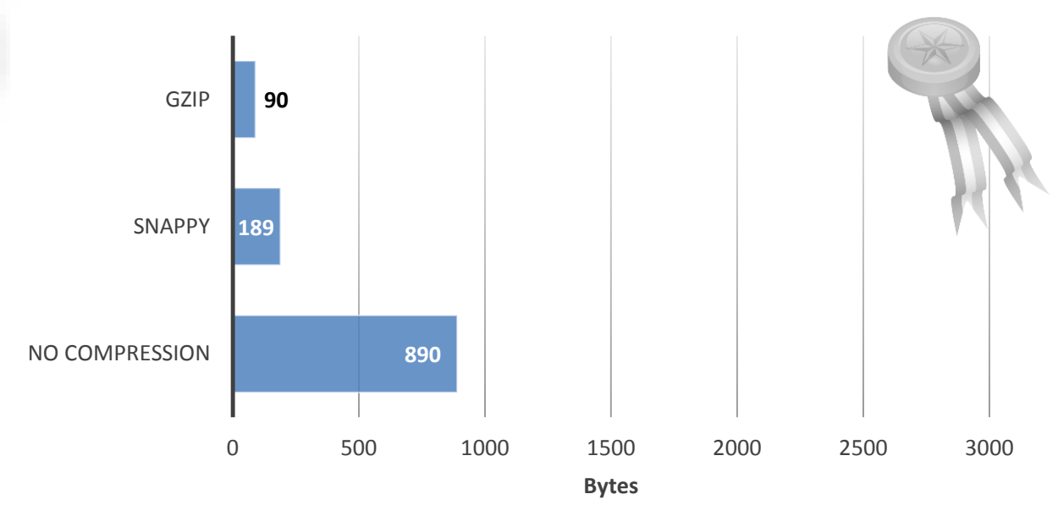
Lookup of given record from Avro file requires to decode and scan fully all the data stored there. This process is parallelized on multiple cores and cluster machines.



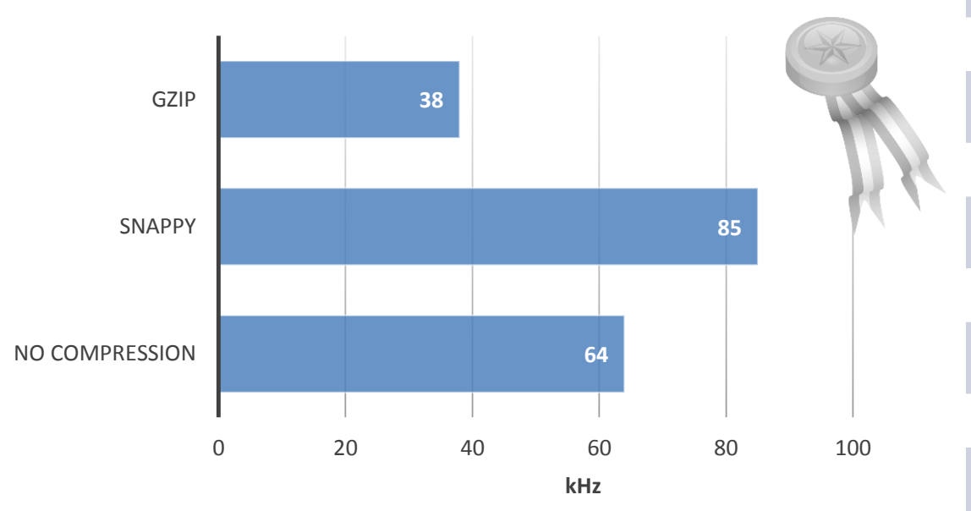
Scanning of Avro files requires decoding of all stored columns. However since the decoding of Avro is lightweight, good scanning performance can be observed.

Apache Parquet file format on HDFS

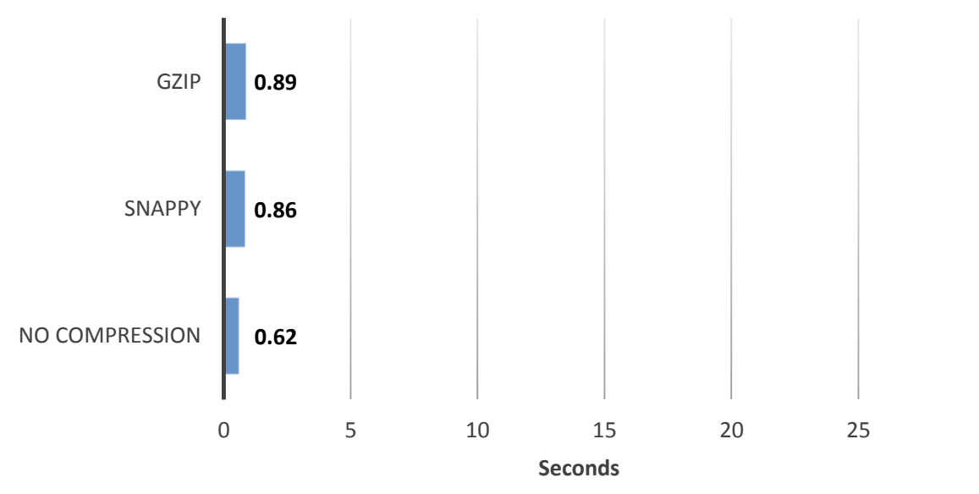
Column oriented data serialization standard for efficient data analytics. Additional optimizations includes encodings (RLE, Dictionary, Bit packing) and compression.



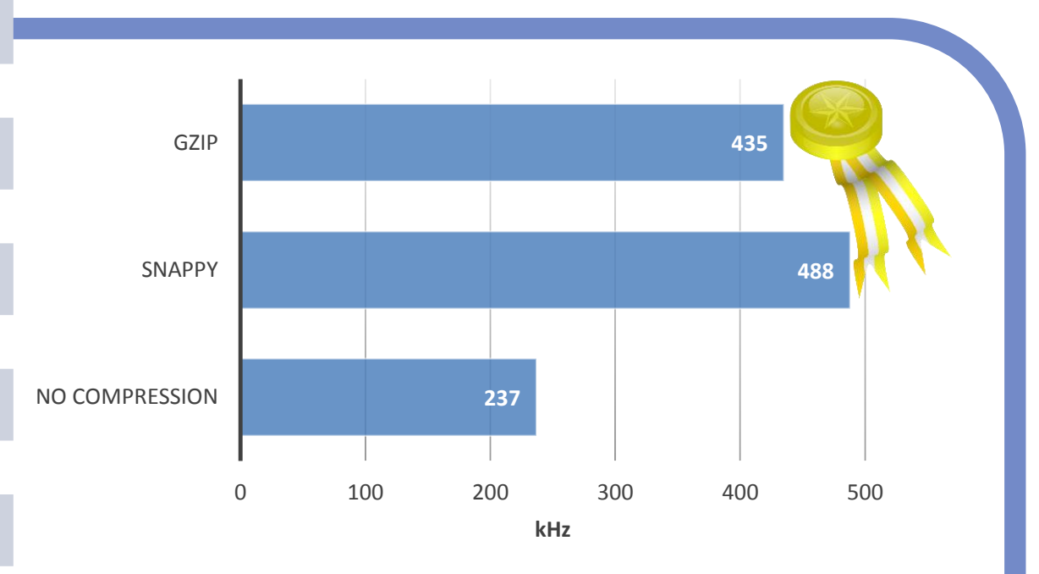
Due to application of encoding and compression algorithms Parquet is space-efficient.



Storing in Parquet format requires building columnar pages in memory and encode the values with various algorithms. Still ingestion performance are very satisfactory.



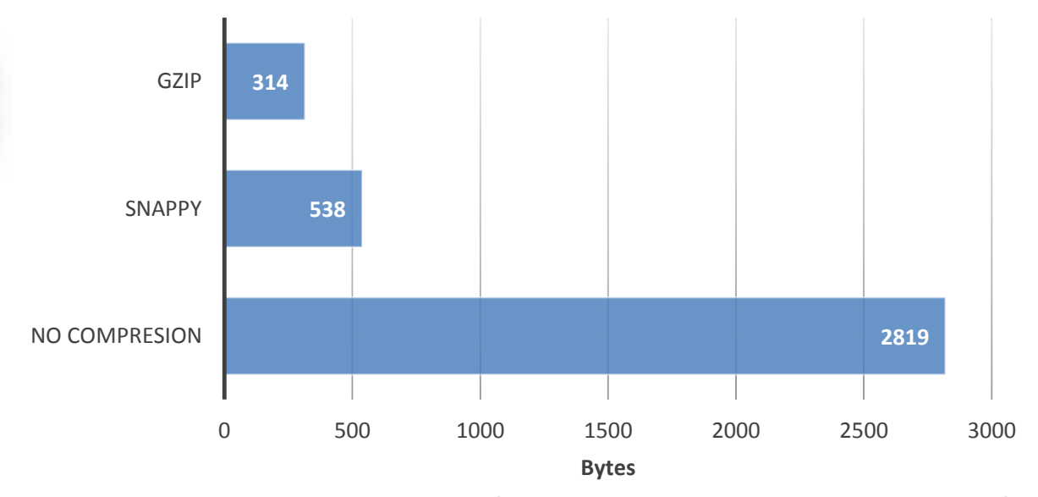
Due to the fact Parquet is a columnar format only the relevant columns have to be accessed. Therefore lookup can be done in an efficient way.



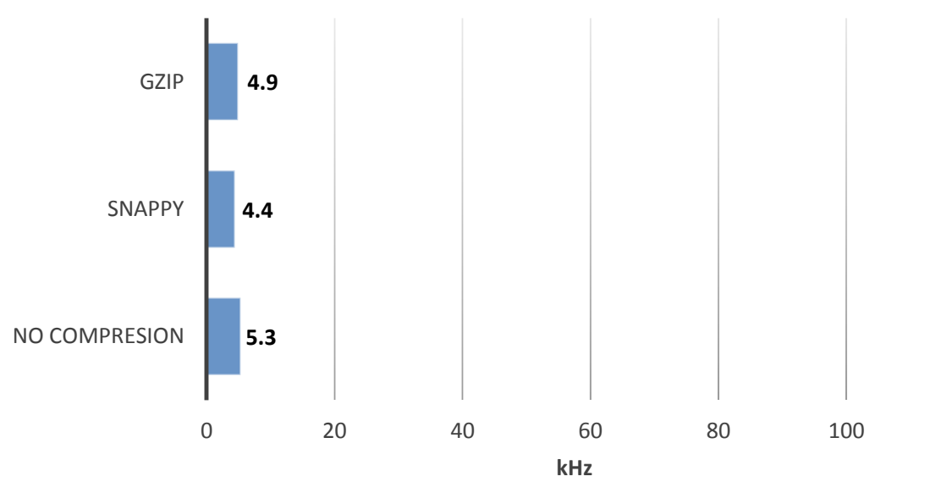
Scanning Parquet files gives very good performance when few out of many columns are being projected as only those will be physically read from HDFS.

Apache HBase storage system

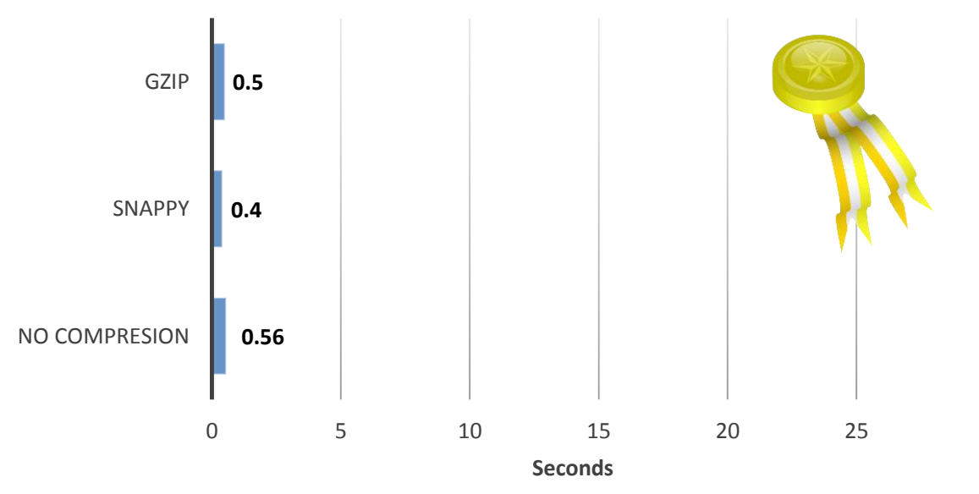
Scalable and distributed NoSQL database on HDFS for storing key-value pairs. Keys are indexed which provides very quick access to the records.



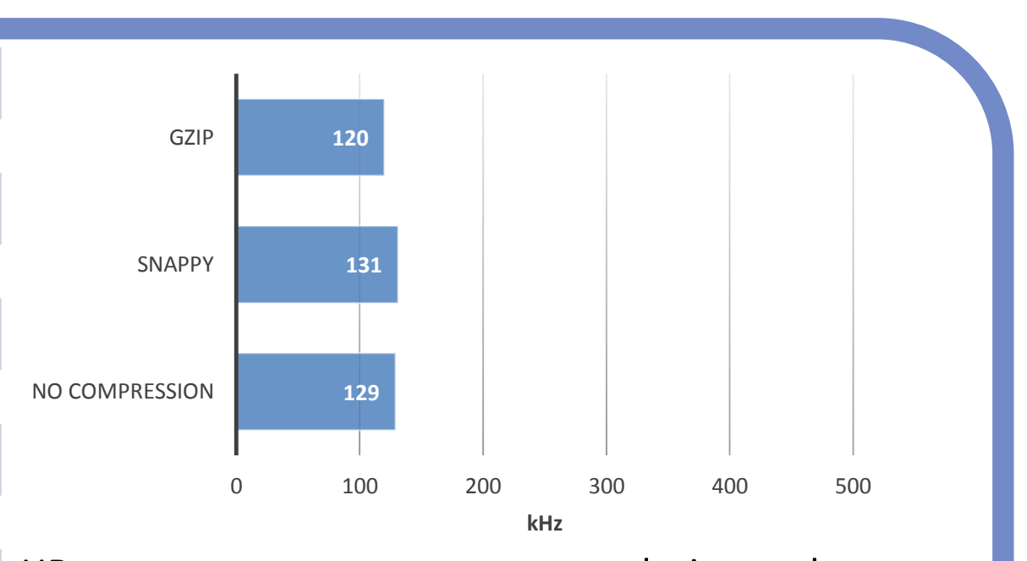
Data stored by HBase by default are not packed – a key of a given row is redundantly stored for each column. It is very important to use compression and block encoding to optimize HBase's use of storage.



Ingestion to HBase is throttled by the need of writing synchronously each operation to a write-ahead log file stored on HDFS.



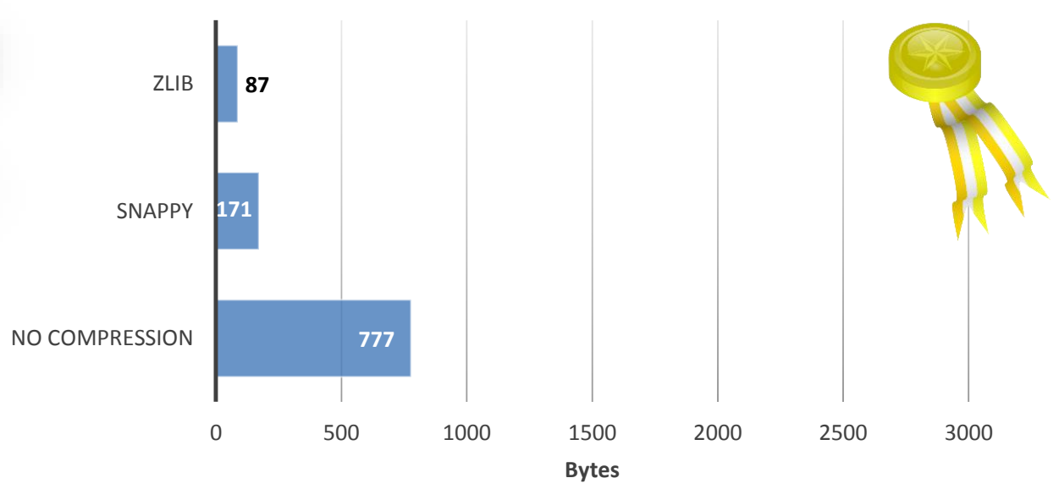
HBase is optimized for data lookup by index. Typically such access takes between 100ms and 1s.



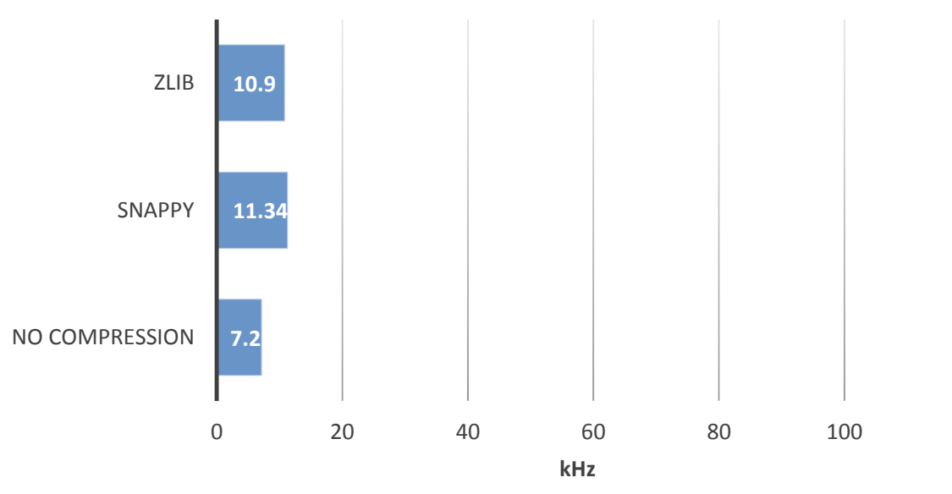
HBase as a storage system was not design to do a performant full processing of data. However after applying several optimization techniques it delivered good scan performance.

Apache Kudu storage system

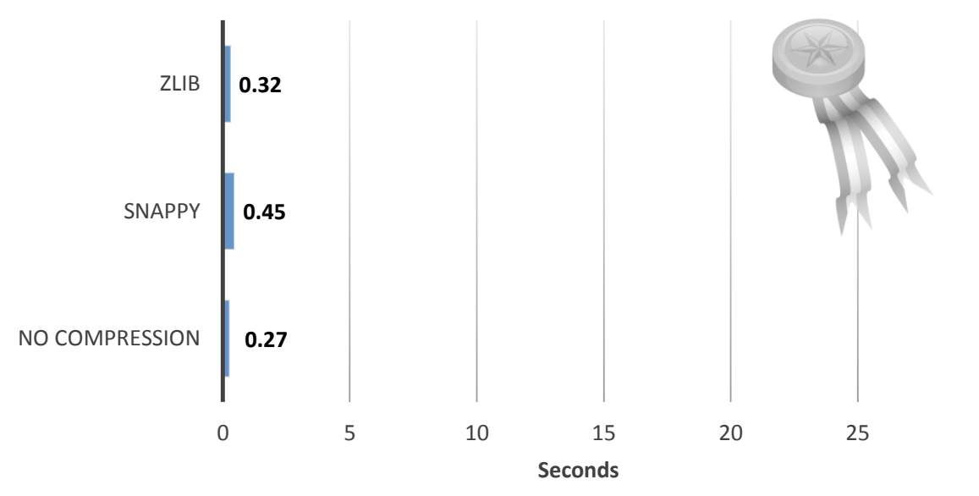
New scalable and distributed table-based storage. Kudu provides indexing and columnar data organization to achieve a good compromise between ingestion speed and analytics performance.



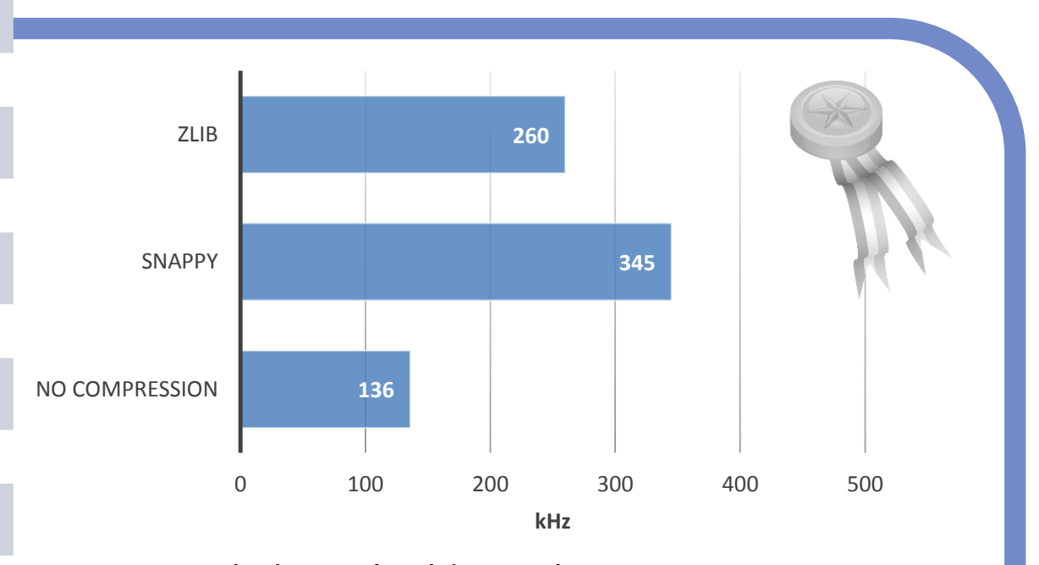
Kudu applies the same encoding algorithms like Parquet.



Kudu like HBase is writing to WAL however it does not relay on HDFS. This reduces latencies to the storage. Additionally the maintenance tasks done in background are less resource demanding.



Similarly to HBase, lookups are very fast because of the internal index. Typically random access is performed between 200ms and 2 seconds.



Because Kudu has a build-in columnar store, it can perform full scans with column projections in a very efficient way.

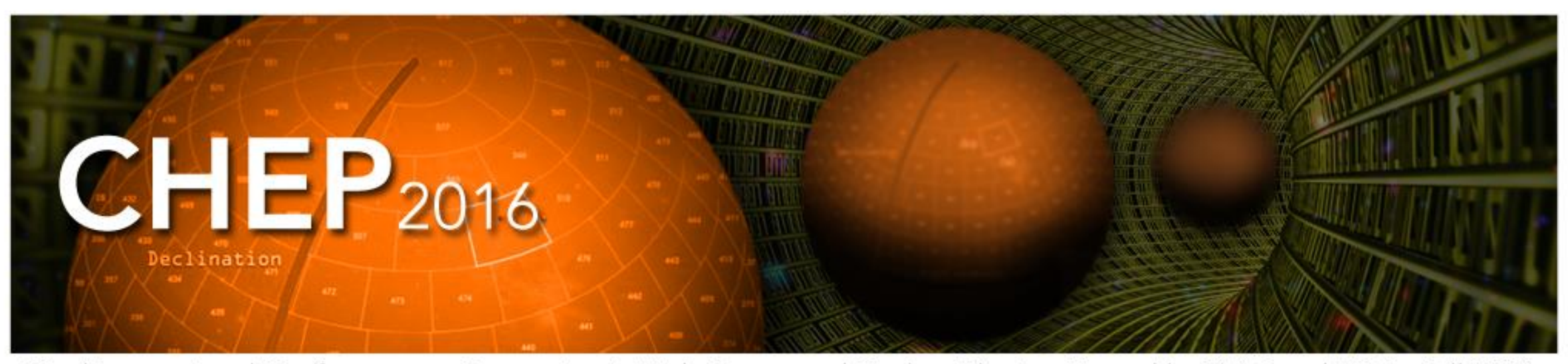
*Event peeking - retrieving global file identification containing the event with provided coordinates (run number, event number,....) is the main use case of the ATLAS EventIndex. Results presented on the plots have been measured by averaging peeking time of various events from different datasets

**Data scanning/counting/reporting is less frequent use case of EventIndex. In the test case the number of events with given trigger mask have being counted across entire data collection.

SUMMARY

Performed evaluation of alternative approaches for storing and accessing data, revealed new opportunities for **improving** ATLAS EventIndex system on:

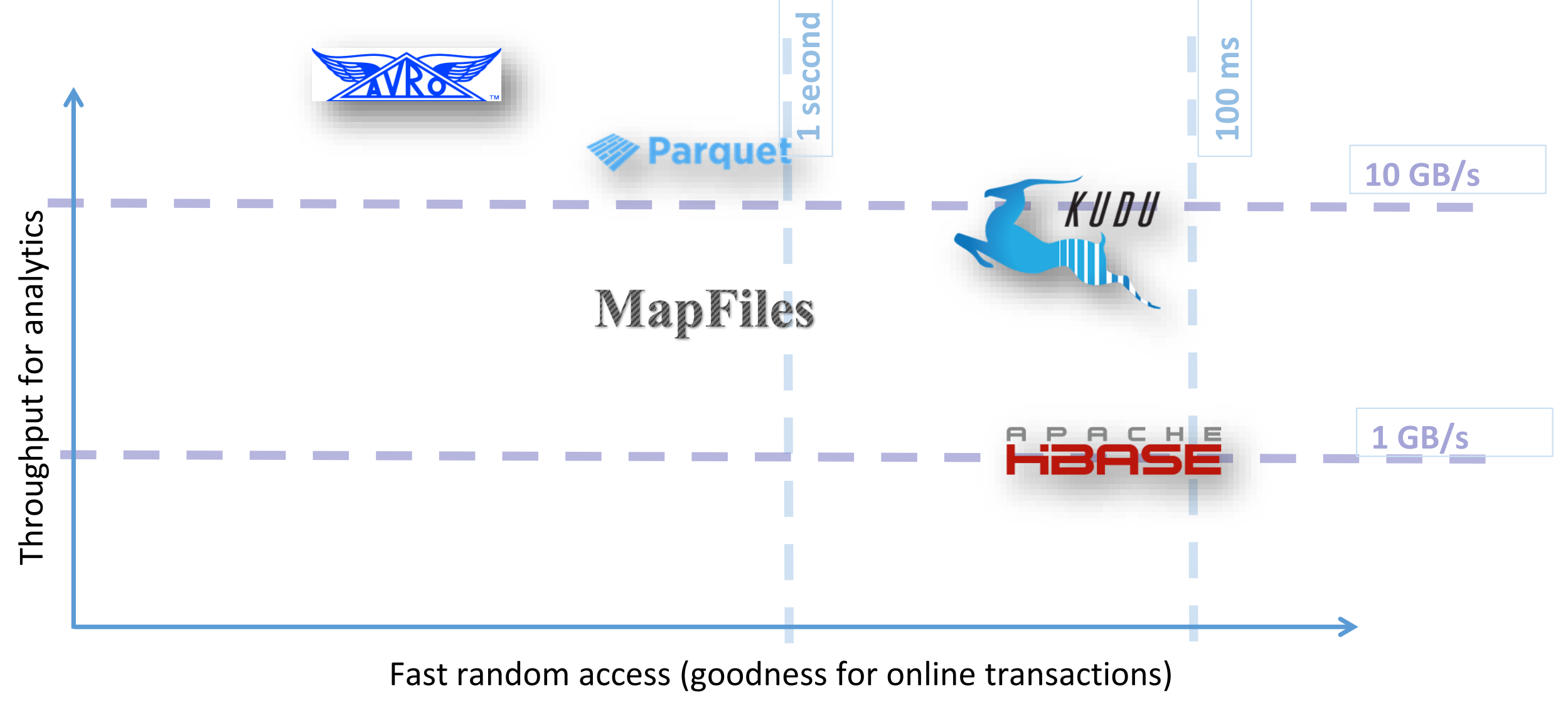
- **Storage efficiency** – with Parquet or Kudu and Snappy compression the total volume of the data can be reduced by factor 10.
- **Data ingestion speed** – all tested solutions provide faster ingestion rate (between x2 and x50) than the current data format used in production.
- **Random data access time** – using HBase or Kudu, typical random data lookup is below 1s.
- **Data analytics** – with Parquet or Kudu it is possible to perform fast and scalable (typically 300k records per second per CPU core) data aggregation, filtering and reporting.
- **Support of data mutation** – HBase and Kudu can modify records (schema and values) in place.



22nd International Conference on Computing in High Energy and Nuclear Physics, Hosted by SLAC and LBNL, Fall 2016

CONCLUSIONS

According to the tests, columnar stores like **Apache Parquet** and **Apache Kudu** achieve the best compromise between **fast** data **ingestion**, fast random data **look-up** and scalable data **analytics**.



Overview of the performance measured with the technologies tested for analytic and random lookup workloads